JIS Interface Server:

Advanced Topics

Version 9.3

September 2026
(originally released January 2005)

his 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.

The name Software GmbH and all Software GmbH product names are either trademarks or registered trademarks of Software GmbH. Other company and product names mentioned herein may be trademarks of their respective owners.

Document ID: JIS-ADV-UG-93-20260923

Table of Contents

About this Guide **21**
Documentation Set 22
Document Conventions 23
Viewing the Documentation Online 24

Part I. About File Management

Chapter 1. Configuration Management Infrastructure in ACE **27**
Application Components 27
Application General Settings Component 28
Library General Settings Component 28
Subapplication Component 28
Subapplication-Specific Files 29
Application Tree Structure 29
Import/Export Location 30
Import/Export Directory Structure 31
Import and Export Operations in ACE 33
Activating the Import and Export Options 34
Export Operation 34
Import Operation 35
Importing Application Components 36
Handling Principal and Dependent Subapplications 37
Considerations When Using Configuration Management 37

Chapter 2. Packaging Applications **39**
Packaging Applications 39
Contents of a Package 40
The Packing Process 40
The Unpacking Process 41
Unpacking KnowledgeBase Files 45
Unpacking a Library without the Main Application 46
Packing and Unpacking KnowledgeBase Files 46
Miscellaneous Remarks 46
Limitations 47

Part II. About Advanced Use of Design

Chapter 3. Date Control **51**
Creating a Date Control 51
Using the Calendar 52
Opening the Calendar 53

Closing the Calendar . . . 53
Navigating the Calendar . . . 53
Changing the Short Date Format . . . 54
Formatting the Date Presentation in the GUI Window . . . 56
Hosts that Change Date Information On-The-Fly . . . 59
DateBaseYear . . . 61

Chapter 4. User Formatting Functions . . . 63
Changes in the Converter.ini File . . . 63
Identifying Pattern Definitions . . . 64
Text Format Definition Dialog Box . . . 65
Using the Text Format Definition Dialog Box . . . 66
Using the Dictionary . . . 66
Calling up a Dictionary . . . 68
Creating a New Dictionary . . . 68
Entering Definitions in a Dictionary . . . 69

Part III. About Window Layouts

Chapter 5. Window Layouts . . . 73
Accessing the Window Layout Definitions Manager . . . 73
Managing Window Layouts . . . 74
Creating New Window Layouts . . . 76
Control Editing Instruction Dialog Box . . . 78
Attaching Window Layouts to Subapplications . . . 79
Window Layout Example . . . 80
Writing Layouts to an *.ini File . . . 81

Chapter 6. Selection Definitions . . . 83
Accessing the Selection Definitions Manager . . . 83
Managing the Selection Definitions List . . . 84
Writing New Selection Definitions . . . 85
Selection Definition Types . . . 86
By Control Type . . . 87
By Host Location . . . 88
Location Limitation . . . 88
Controls Specification . . . 89
By Screen Section . . . 90
By Primary Pattern Definition . . . 93
By Pattern Definition . . . 94
By Representation Definition . . . 94
Multi-Parameter Selection Definitions . . . 95
Other Than the Selection Definition . . . 97

Chapter 7. Function Definitions 99
Accessing the Function Definitions Manager 99
Managing Function Definitions. 100
Writing New Function Definitions 101
Function Definition Types 102
Anchor controls. 103
The Distance Section 104
The Advanced Measurement Dialog Box 105
Equal Size 107
Size Each 109
Adjust Size By Text 110
Place 110
Place Each 113
Place Each Relative To the Display 114
Center 115
Center Each 116
Align 117
Alignment Type Section 117
Align To 118
Equal Spacing 119
Spacing Type. 119
Horizontal and Vertical Spacing 120
Spacing 123
Set Font and Color 124
Add Representation 124
Apply Window Layout 126
Group of Functions 129

Chapter 8. Dynamic Sections 133
How Dynamic Sections Work 133
Applying Dynamic Sections 134
Defining Dynamic Sections in Layout View. 134
Dynamic Areas in Runtime. 136

Part IV. About Enhancing Your Application's Usability

Chapter 9. Tab Controls 141
Creating Tabs for Subapplications. 141
Starting the Create Tab Wizard 141
Creating Tabs 142
Completing the Wizard 144
Rearranging the Tabs. 144
Changing Tab Style Options 145
Editing Tabs 145
Deleting a Tab control. 146
Renaming a Tab. 146
Moving Controls from One Tab to the Other 146

Chapter 10. Many-to-One 147
The Purpose of Many-to-One 147
An Overview of the Many-to-One Workflow 148
The Many-to-One Workflow 149
Creating Dependent Subapplications 149
Creating the Principal Subapplication 150
Defining a Subapplication as Principal 150
Analyzing the Dependents and the Principal 152
Analyzing the Dependents 152
Analyzing the Principal 153
Example of Dependents and Principal Analysis 153
Representing Dependents as Tabs on the Principal 155
Listing Transitions in the <ApplName>.ini file 156
Runtime Operation of a Many-to-One 156
Transition Information 157
Transition Modes 157
The UserMoveToDependentScreen Method 158
Writing Transition Information in <ApplName>.ini Using Host Keys 158
The ValidateDependentScreenByName DoMethod 162
ValidateDependentScreenByName to Modify Enter Method 163
Message Handling 164
Important Points and Limitations to Remember 164

Chapter 11. One-to-Many 167
The Way One-to-Many Works 167
Subwindows 168
Creating a Subwindow Using Add Control 168
Creating a Subwindow Using the Definitions Palette 172
Creating a Subwindow Through the KnowledgeBase 172
Moving Elements into a SubWindow 172
Enabling Display of an Initially Hidden Subwindow 173
Attaching Methods 173

Chapter 12. Skipping Windows 177
Skipping Window-Display in Runtime 177
Improving Performance Time 178
Never Display Window 179
Conditionally Display Window 179
System-Triggered Methods With Skipping Window–Display 179
UserSkipSubApplication 179
UserShouldWindowBeBuilt 180
Auxiliary Methods Used With Skipping Window-Display 181
Example Use of the Methods 182
Never Display the Window in Runtime 183
Setting the Subapplication Type. 184
The UserSkipSubApplication Method 184
Conditionally Display the Window in Runtime 185

Condition is Host Screen Independent 185
Condition is Host Screen Dependent 187

Part V. About Special Host Application Requirements

Chapter 13. Using Just-in-Time GUI Subapplications 191
Just-in-Time GUI and Unrecognized Screens 191
Other Uses of the Just-in-Time Mechanism 192
Creating Additional JIT Type Subapplications 192
The NO_ATTRS JIT Type Subapplication 193

Chapter 14. Application-Specific Configurations 195
Creating Multi-Line Text Boxes 195
Designating a Text Box as Multi-line 196
Writing an Update Method for Multi-line Text Boxes 196
Allowing the Maximum Number of Characters 197
Support for the Ampersand (&) Character 198

Chapter 15. Message Handling 199
Message Definition Types 199
Messages Definitions View 199
Designating a Pattern Definition as a Message Definition 200
Message Handling Methods 200
Defining a Message Handling Method 201
Message Display 203
Creating Multi-line Message Boxes 203
Setting Up Your System to Handle Multi-line Messages 204
Message Waiting Indicator 204
Handling the Message Waiting Indicator 204
Message Help 204

Chapter 16. Handling Popup Windows 207
How ACE Identifies a Host Popup Window 207
Creating Popup Windows With the Wizard 208
Controlling Popup Display in Runtime 209
Processing a Popup Window Subapplication 210
Layouts 210
Batch Mode 210
Message Handling 210
Runtime Screen Identification 210
Displaying the Popup in Runtime 211

Chapter 17. Lists . . . 213
The List Structure . . . 214
Working with the List Wizard . . . 215
Recognizing a List . . . 215
List Analysis Process . . . 216
Finding the List Area . . . 216
Finding Columns, Headers and Linking Headers to Columns . . . 217
Finding the Header and Column Values . . . 217
List Type Pattern Definition Structure . . . 218
The Pattern Definition Structure of List with Parameters . . . 218
Argument Description . . . 219
Placement . . . 220
ListLines . . . 220
Restricting the Influence of ListLines . . . 220
Placement With Screen Captures . . . 221
Placement Includes List Header . . . 221
Number of Header Lines . . . 222
Number of Lines Between Headers and List . . . 222
Placement With DDS Panels . . . 222
Searched Columns . . . 223
Non DDS Applications . . . 223
DDS Applications . . . 224
List Headers . . . 225
How the Analysis Works . . . 225
Certain Headers . . . 226
Hard Delimiter . . . 226
Soft Delimiter . . . 227
Using Certain Headers to Join Header Elements . . . 227
Linking Headers to Columns . . . 227
Hard Delimiter Argument . . . 228
Soft Delimiter Argument . . . 228
Linking the Headers . . . 229
Certain Headers Parameter . . . 229
List Columns . . . 230
Pattern Definitions of List Column Type . . . 230
The Column Argument's Pattern Definition . . . 231
The Header Argument's Pattern Definition . . . 231
Arranging List Columns and Headers Manually . . . 231
Viewing the Links Between the Headers and the Columns . . . 232
Links Between the Headers and the Columns . . . 233
Linking Headers and Columns . . . 233
Removing Links between Headers and Columns . . . 234
Deleting a List Column or a List Header . . . 234
Declaring an Area as a List Header or a List Column . . . 234
Creating New Columns to the Right . . . 235
Modifying the Width of a Header or a Column . . . 235
Location . . . 236
Dimensions . . . 236

Splitting a List Header 236
Splitting a List Column 237
Extending a List Column 239
Deleting a Column 239
Modifying the Type of a Column 239
Types of Lists 240
Standard Lists 240
Folded Lists 241
KnowledgeBase Modifications 241
Modifying a Folded List's Parameters 242
Repeated Columns Lists 243
Creating a Pattern Definition for a Repeated Columns List 243
List With Parameters Type Pattern Definitions 245
List Pattern Definitions 245
Folded Lists 246
Working with Sections 248
Adjusting List Sections in Panels not From DDS 248
Adjusting List Sections in Panels from DDS 248
Processing Lists Through the Views 249
Lists and Design View 249
List With Parameters 249
Appearance and Functionality of the Table Control 249
Selecting Rows and Columns 250
The Table Component Style Tab 251
Row Properties/Column Properties 254
Making a Local Modification 255
Resizing Columns and Rows 255
Changing Column Order 255
Modifying the Properties of Individual Columns 255
Lists and Methods 256
Lists and Runtime Field Information View 257
Top and Bottom Definition 258
Fold Definition 258
Retrieval Buffering 259
List Paging Mechanism 260
Caching Host List Pages to Create Multi-Page Tables 261
Caching Number of Host Pages to a Single Table Display 262
Caching Multiple Sets of Host List Pages 262
Using a Method to Change the Number of Cached Host Pages 263
Lists and Runtime Screen Identification View 263

Chapter 18. External Data 265
External Data Source Connectivity 265
External Data Methods 265
ExternalData DoMethods 266
DBSession Connectivity 267
Initial Settings for DB Sessions 268
DBSession DoMethods 269

Creating a DBSession Method 271
Retrieving Data 271
Manipulating the Retrieved Data 272
Inserting Data into the Database 272
Retrieving a Previously Allocated DB Session 272
HostSession Connectivity 273
HostSession DoMethods 273
Creating a HostSession Method 276
Navigating Through the Application 276
Inserting and Retrieving Data 276
Retrieving a Previously Allocated Host Session 277
Troubleshooting ExternalData Methods 277

Part VI. About Using *.ini Files to Increase Efficiency

Chapter 19. Using *.ini Files 281
Using Multiple *.ini Files 281
Multiple *.ini Files 281
Loading Files into the Cache Memory 282
Updating the Cache Dynamically in Runtime 282
Using DDS.ini and SDF.ini 283
Using the DDS.ini 283
Editing the DDS.ini 283
Using the SDF.ini 285
Forcing a Field’s Contents by Name 285
Forcing a Field’s Contents by Place 285
Editing the SDF.ini 285
Entries in <ApplName>.ini 286
<ApplName>.ini 286
The Program Section of <ApplName>.ini 287
Entries Governing Prompt Controls 290
Entries Governing Cursor Selection Menus 292
The Emulator Section of <ApplName>.ini 293
The Class Size Section of <ApplName>.ini 295
The Window Layout Section of <ApplName>.ini 296
The Control Editor Section of <ApplName>.ini 296
The Fix <EmulatorName> Section of <ApplName>.ini 297
Converter.ini File 299
*.ini File Settings 299
Entering Values for Windows Controls in the *.ini File 300
Set ACE to Read the Values From the *.ini File 300
Updating a Combo Box from the *.ini File 302
Entering Combo Box Items from the *.ini File 302
Format Enhancement to Updating the ComboBox 304
Reading Check Box Values from the *.ini File 305
Saving the Last Check Box Setting 306
Creating a Logical Name for the Check Box 307

Translating Accelerator Keys 307

Part VII. About JI Integration Methods in JIS Interface Server

Chapter 20. Invoking JI Integration Methods from JIS Interface Server 311
When to Use This Feature 311
Software Requirements 312
When Not to Use This Feature... 312
Activation 312
Workflow 313
Design Time 313
Runtime Generation 314
Runtime 314
Design Time - Writing the ACE Method 315
JI Integration Map-List-Map Data Model 315
Creating Structure and Array Objects in ACE Methods 317
Syntax for Specifying the Path of an Object 318
Method Line Types 318
DoMethods for Invoking a JI Integration Method 321
DoMethods Executed by Structures and Arrays 323

Part VIII. About Customizing the ACE Menu Bar

Chapter 21. Adding Items to the ACE Menu Bar 329
Overview 329
Defining User Items on the ACE Menu Bar 330
The XML File 330
Tags in the user_definitions.xml file 333
Parameters for Use Within the ACTION Tag 335
*.ini File Settings 339
Scope of *.ini File Settings 339
Parameters 339

List of Figures

Application tree structure 30
General Options dialog box 30
Structure of the import/export directory 32
Selection operations in import/export directory 33
The Import and Export menu options 34
Export Application Components dialog box 34
Import Application Components dialog box 35
Specifying the libraries and subapplications to unpack 42
Add/Remove subapplications from selection dialog box 42
How to select subapplications without name restrictions 43
Specify Target Application step 43
Directory and file tree 44
Confirm File Replace message box 45
Add Control dialog box 52
Calendar control 53
Regional Options dialog box 55
FormatDate dialog box 56
Assigning more than one value to a date control 57
Formatting more than one parameter 57
Format Date Parameter dialog box 58
Available Date Formats list 59
Pattern Definition Properties dialog box 64
User format feature disabled warning message 64
Available and Selected lists in the Text Format Definition dialog box 65
List of Dictionaries dialog box 68
List of Dictionaries dialog box - New Dictionary field 69
Word Dictionary dialog box 70
Window Layouts menu item in ACE 74
Window Layout Definitions manager 74
Instructions list area of the Window Layout Definition dialog box 76
Control Editing Instruction dialog box 78
Layouts dialog box 79
A subapplication derived from an analyzed screen image 80
A new Window Layout definition 80
A new subapplication after the application of a new Window Layout 81
Accessing Selection Definitions through the Design menu in ACE 83
Selection Definitions manager 84
Choosing the Selection Definition type 85
By Control Type dialog box 87
By Host Location 88
Location limitations of a host screen 89
Control specification drop down list 89
By Screen Section 91

Multi-parameter selection definitions 95
Selection Definition dialog box. 96
Other than the selection definition option 97
Function Definitions manager 100
Choose a function definition type 101
Distance section. 105
Advanced Measurement dialog box 105
Advanced measurement options 106
Advanced measurement - percent of width of. 106
Naming the function definition 107
Function definition - Equal Size 107
Function Definition - Place options. 111
Align-to section 118
Order option. 120
In Rectangle option 120
Horizontal and vertical spacing options 120
Set font and color. 124
Chosen window layout list 126
Group of functions. 129
Host screens with dynamic patterns 134
Apply Screen Layout dialog box. 135
Dynamic Input/Output Periods section 135
Pattern definition for a dynamic area 136
Create Tab wizard 142
Tab style options 145
Many-to-One 148
Principal and dependent subapplications. 149
Creating a Dependent subapplication 150
Creating a Principal subapplication. 151
Principal Parameters dialog box 151
Screen 1 153
Screen 2 154
Create tabs for subwindows check box 155
Displaying two dependent subapplications as tabs 156
A principal and three dependants 160
When transition modes must be included 162
ValidateDependentScreenByName DoMethod. 163
Add Control dialog box. 168
Style tab of the SubWindow component. 169
Manager tab of the SubWindow component. 169
Events tab of the Button component dialog box 174
Define Method dialog box. 174
Three phases of a regular subapplication during runtime 178
Distribution menu 183
Setting the subapplication type in the New Subapplication wizard 184
Subapplication Properties dialog box 184
Just-in-time GUI check box in the New Subapplication wizard. 192
Creating multi line text boxes 195

Style tab of text box component . . . 196
Method that updates host with data in the specified multi-line . . . 197
Input field in Analysis view . . . 198
Input field in Design view . . . 198
Decomposition Definition Properties dialog box . . . 198
Message Definitions View icon . . . 200
Message Handling dialog box . . . 201
A host popup identified by its unique border . . . 208
Identifying pop-ups in the New Subapplication window . . . 209
Window popup field in Subapplication Properties dialog box . . . 209
List elements . . . 214
List in host view . . . 215
The way ACE analyzes a list . . . 216
List area . . . 216
Links between headers and columns . . . 217
List header and column values . . . 217
Replacing list arguments . . . 218
ListLines . . . 220
Placement with screen captures . . . 221
List parameters tab . . . 221
Number of lines between headers and list . . . 222
List filters and the placement pattern definition . . . 223
ListColumnsSearchedRegular . . . 224
Verify SDF columns using Searched Columns definition check box . . . 224
How the analysis works . . . 225
The Certain Headers argument . . . 226
Hard delimiters . . . 226
Soft delimiters . . . 227
Using certain headers to join header elements . . . 227
Hard delimiter argument . . . 228
Hard delimiter example . . . 228
Soft delimiter argument . . . 228
Certain Headers parameter . . . 229
ListColumnComboBoxViaINI . . . 231
ColumnHeaderTable . . . 231
List headers warning . . . 232
Links between headers and columns . . . 233
Result of a relationship between header and column not being found . . . 233
Result of a relationship between header and column being found . . . 234
Modifying the width of a header or a column . . . 235
First and last name details are included in one column . . . 237
Problems in List analysis that can be fixed using the List features . . . 238
Deleting an extra column . . . 239
Folded lists in host view . . . 241
Modifying a folded list’s parameters in the Parameters tab . . . 242
Setting the number of list header and separation lines . . . 247
Placement of list sections . . . 248
A table as it appears during runtime . . . 249

Table component's Style tab 251
Row and Column Properties dialog box 254
Decomposition Definition Properties dialog box 257
List Properties dialog box 258
Retrieval Buffering options 259
List paging mechanism options 260
Window Components palette 300
Manager tab in the Component dialog box 301
Combo box during runtime 304
GetAllLoanInformation method 317
Define Method dialog box 318
CreateArray dialog box 319
CreateStructure dialog box 320
The default ACE menu bar 329
The ACE menu bar, modified 329
Example of a user defined menu structure 329
The example menu structures (1 of 2) 332
The example menu structures (2 of 2) 332

List of Tables

JIS Interface Server documentation set 22
Documentation conventions 23
Subapplication specific files 29
Contents of each directory under import/export 32
Changing the date 53
Text Format Definition dialog box buttons 65
Text Format Definition dialog box fields 66
Dictionary options 66
Window Layout Definitions manager buttons 75
Window Layout Definition dialog box buttons 76
Control Editing Instruction dialog box buttons 78
Selection Definitions manager buttons 84
Selection Definition types 86
Location limitations of a host screen 89
Control specification drop down list options 90
Selection Definition dialog box options 96
Function Definitions manager options 100
Function Definitions manager options 102
Distance section options 104
Function definition - Adjust size by text options 110
Function definition - Place Each Relative To Display options 114
Function definition - Align Type option 117
Horizontal and vertical spacing options 121
Margin options 121
Chosen function definition section options 130
Three transition modes 157
Valid host actions 159
How to write transitions when there is a succession of actions 161
Window styles 170
Initial state in runtime 171
Dialog styles 171
WriteUserVariable 181
GetUserVariable 181
DeleteUserVariable 182
Message Handling dialog box options 201
List elements 214
List arguments 219
Parameters tab options 242
Header Location combo box options 242
List With Parameters type pattern definitions 245
Ways the header row of a folded list can be arranged 246
Table flags options 251
Horizontal Scrollbar options 252

Vertical Scrollbar options 252
Selection options 253
Basic Styles options 253
Row and Column Property options 254
List methods 256
Retrieval Buffering options 259
ExternalData DoMethods 266
DBSession DoMethods 269
HostSession DoMethods 274
Program section of the <ApplName>.ini 287
Prompt keys and values 290
Entries governing cursor selection menus 292
Emulator section of the <ApplName>.ini 293
Class Size section of the <ApplName>.ini file 295
Window Layout section of the <ApplName>.ini 296
Control Editor section of the <ApplName>.ini 296
Fix <EmulatorName> Section of the <ApplName>.ini 297
Converter.ini file settings 299
Application and subapplication settings 301
Entries for example combo box and their purpose 302
Steps to be performed in design time 313
Steps to be performed in runtime generation 314
Steps to be performed in runtime 314
Transaction data 316
ConnectToJIService 322
InvokeJIMethod 322
GetHostSessionName 323
CloseHostSession 323
GetArray 324
GetString 324
GetStructure 324
ContainsObject 325
CopyObjects 325
RemoveObjects 325
SetObjects 326
Equals 326
GetSize 326
Tags in the user_definitions.xml file 333
List of predefined parameters for use in user menus 336

List of Examples

Application tree sructure 30
Import/Export directory structure 31
Selection operations in import/export directory 33
Export operation 34
Import Operation 35
Hosts that change date information on-the-fly 60
By screen section 92
By primary pattern definition 93
By representation definition 95
Other than the selection definition. 97
Advanced measurement. 106
Equal Size 108
Function definition - Place option 111
Function definition - Place Each option 113
Function definition - Center option 116
Function definition - Center Each option 116
Function definition - Align-to option 118
Horizontal and vertical spacing 122
Spacing 123
Add representation. 124
Nested window layouts 127
Group of functions. 130
Displaying two dependent subapplications as tabs 156
Transition section in the <ApplName>.ini file 160
How to write transitions when there is a succession of actions. 161
When transition modes must be included 161
ValidateDependentScreenByName DoMethod. 163
Specific.ini file JITGUI setting 193
Allowing the maximum number of characters 197
Message handling methods 201
Lists 215
Number of lines between headers and list. 222
Hard delimiters 228
Certain Headers 229
ListColumnComboBoxViaINI 230
Problems in List analysis that can be fixed using the List features. 238
Folded lists. 241
Headers and separation lines 247
Caching host pages in tables 262
Caching multiple sets of host list pages. 262
Querying an iSeries DB2 database 268
Opening a DB session with parameters: 272
<ApplName>.ini 286

Entering combo box items from the *.ini file 302
Format enhancement to updating the combo box 305
Reading check box values from the *.ini file 306
JI Integration Map-List-Map data model 316
Syntax for specifying the path of an object 318

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. In JIS 9.3, application and library metadata use SQLite. Pack/Unpack and library import handle eligible BDE database migration while retaining the original data after successful conversion. Install JDK 21 and a signed XML license before upgrading. Use the JIS 9.3 Release Notes for installation, runtime packaging, and compatibility requirements.

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.

The manual discusses the advanced features of ACE. It is divided into the following sections:

- Part I - "About File Management"
- Part II - "About Advanced Use of Design"
- Part III - "About Window Layouts"
- Part IV - "About Enhancing Your Application's Usability"
- Part V - "About Special Host Application Requirements"
- Part VI - "About Using *.ini Files to Increase Efficiency"
- Part VII - "About JI Integration Methods in JIS Interface Server"
- Part VIII - "About Customizing the ACE Menu Bar"

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.

Part I. About File Management

This section describes two ACE utilities that enable multiple developers to work on the same set of files: Configuration Management and Packing/Unpacking.

When more than one developer is working on the same ACE Application, there is often a need to transfer parts of the Application between developers. Configuration Management allows you to manage development by importing and exporting components to and from ACE. The hierarchical structure of the ACE Application allows you to import and export specific components of the Application for other developers to work on. Components move to and from an import/export directory where they can be accessed by other developers.

ACE's Pack utility is used to compress Application files. This facilitates the transfer of application files via e-mail and saves space when creating archives. Applications packed on one computer can then be unpacked on other computers enabling multiple developers to work on the same Application files.

Chapter 1. Configuration Management Infrastructure in ACE

JIS Interface Server provides an infrastructure in ACE that enables you to manage your development environment using a configuration management tool. The infrastructure is based on the ability to export Application components from ACE to a specific directory, and then import these components back into ACE. The directory which serves as the import/export location is used to interact with the chosen configuration management tool.

This chapter contains the following topics:

- Application Components
- Application Tree Structure
- Import and Export Operations in ACE
- Handling Principal and Dependent Subapplications
- Considerations When Using Configuration Management

Application Components

In order to import or export parts of an ACE Application, it is necessary to be able to package each part separately.

For this purpose, the following three components have been defined:

- Application general settings
- Library general settings
- Subapplication

These components reflect the hierarchical structure of an ACE Application. An ACE Application may contain one or more libraries. A library contains one or more Subapplications. The Application itself may also contain one or more Subapplications and functions as a library in this respect.

Each component is a compressed binary file, which can be placed in the import/export directory, and then checked into/out of the configuration management tool. The binary file includes all the information pertinent to the component. This information may include entire files as is, as well as configuration data.

Application General Settings Component

The Application general settings are global settings for the entire Application, such as information contained in database files and in *.ini files. For example, this information includes Application Level Methods from the APPLICAT.ION file.

The file containing the Application general settings is named

`<ApplName>.jpa.`

Library General Settings Component

The library general settings include information such as KnowledgeBase files, *.ini files, and database records. For example, this information includes library methods from the LIBRARY.ION file and global settings from the library's SPECIFIC.INI file.

The main Application in ACE may also contain Subapplications, and functions as a library in this respect. Therefore, the library general settings are also saved for the main Application.

The file containing the library general settings is named

`<LibName>.jpl.`

Subapplication Component

The Subapplication component contains Subapplication-specific data, which includes Subapplication-specific files, *.ini file settings, and database entries. The Subapplication component also contains the input files that created the Subapplication, such as *.pnl, *.ddo, *.sdf files.

The file containing the Subapplication information is named

`<SubApplName>.jps`.

Subapplication-Specific Files

The Subapplication-specific files located in the library or Application directory are:

Table 3. Subapplication specific files

File Name	File Format	File Content
*.acc	Text	Subapplication accelerator information.
*.gad	Binary	General Subapplication information, such as analysis information.
*.glm	Text	Local modifications performed on a Subapplication in Design View. If no local modifications were performed on a specific Subapplication, this file is not created.
*.mnu	Text	Subapplication menu information.
*.sa	Text	Subapplication specific methods.

Application Tree Structure

In order to select the components to export or import, an Application is displayed in a tree structure.

The root of the tree is the Application name. An Application contains a subdirectory with the Application general settings, and below that, a subdirectory for each one of the Application libraries, including the Application itself, which functions also as a library. The Application itself is always the first library listed, followed by the rest of the libraries in alphabetical order.

Each library has two subdirectories. The first contains the library general settings. The second, named Subapplications, contains a list of all the Subapplications in the library. When a specific library does not contain any Subapplications, its Subapplications directory in the tree is disabled.

Example 1. Application tree sructure

The following tree illustrates the structure of an Application named EXAMPLE. This Application contains several Subapplications, and also has two libraries - EXLIB1 and EXLIB2. The library EXLIB2 does not contain any Subapplications, and therefore the Subapplications directory is disabled.

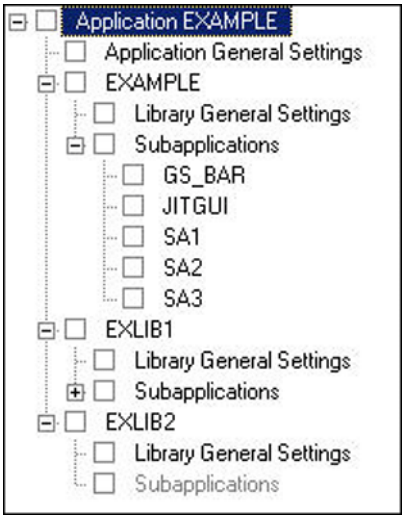


Figure 1. Application tree structure

Import/Export Location

The ability to export Application components from ACE, and then import them back into ACE, requires a location in the file system that can be used by these operations.This location, referred to as the import/export directory, is configured in ACE.

To configure an Import/Export directory:

1 From the *Options* menu select *General Options*.
The following dialog box opens:

Figure 2. General Options dialog box

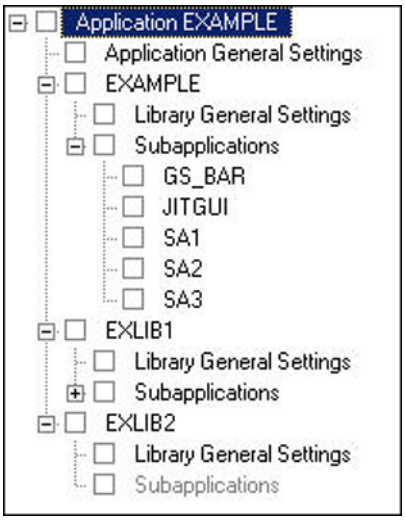
2. *Import/Export directory* - Specify the directory to which files are exported and from which files are imported.
   The default directory is shown above. The characters ./ indicate that the directory is relative to the ACE working directory. For example, if ACE is installed in C:/ACE, then the default import/export directory is C:/ACE/ImportExport.
3. *Delete files after importing?* - Specify whether to delete files from the directory after importing them. The default, shown above, is to always ask the ACE developer.
4. *Log File* - Specify the name and path of the log file for the import/export operations. The default, shown above, is a file named importexport.log located in the ACE working directory.
5. Click *OK* to save your settings.

Import/Export Directory Structure

The import/export directory structure resembles the tree structure of an Application.

Example 2. Import/Export directory structure

For an Application with the following tree structure:



This is the structure of the import/export directory:

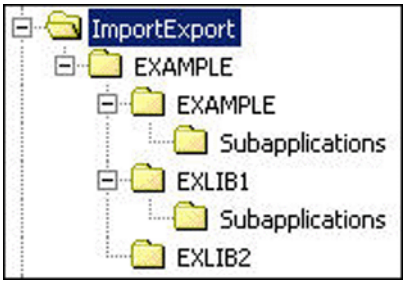


Figure 3. Structure of the import/export directory

These are the contents of each directory under the ImportExport directory:

Table 4. Contents of each directory under import/export

Directory	Files	Description
EXAMPLE	EXAMPLE.jpa	Application general settings.
EXAMPLE/ EXAMPLE	EXAMPLE.jpl	Library general settings.
EXAMPLE/ EXAMPLE/ Subapplications	GS_BAR.jps JITGUI.jps SA1.jps SA2.jps SA3.jps	A Subapplication-specific package for each Subapplication in the EXAMPLE Application.
EXAMPLE/ EXLIB1	EXLIB1.jpl	Library general settings.
EXAMPLE/ EXLIB1/ Subapplications	LIB1SA1.jps LIB1SA2.jps	A Subapplication-specific package for each Subapplication in the EXLIB1 library.
EXAMPLE/ EXLIB2	EXLIB2.jpl	Library general settings.

Import and Export Operations in ACE

The import and export operations in ACE are performed using an Application tree dialog box which displays the hierarchical structure of the Application you are importing files to or exporting files from.

You use the tree to select the Application components to import or export. The tree is composed of parent and child nodes. Parent nodes, such as an Application or a library, contain child nodes. Child nodes, such as a Subapplication package, do not contain any other nodes.

Each node in the tree has a check box, which enables you to select it. When selecting nodes in the tree, the following rules apply:

- Selecting or clearing a parent node, automatically selects or clears all its child nodes.
- Selecting one or more child nodes, but not all the child nodes, causes the parent node's check box to be checked with a gray background.

Example 3. Selection operations in import/export directory

In the tree below, the EXLIB1 library node was selected. This caused both its child nodes, Library General Settings and Subapplications, to be selected. In addition, the Application EXAMPLE node's check box is checked with a gray background, because not all of its child nodes are selected.

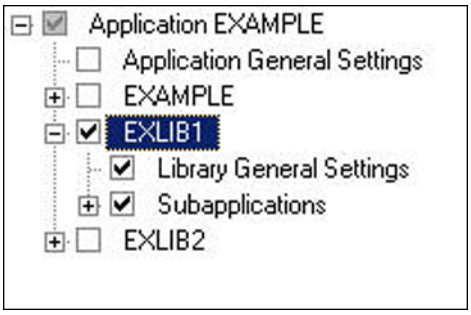


Figure 4. Selection operations in import/export directory

Activating the Import and Export Options

The *Import* and *Export* options are available from the *Utility* menu in ACE:

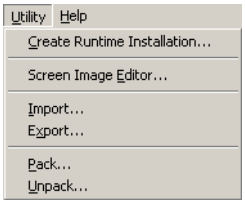


Figure 5. The Import and Export menu options

Export Operation

The *Export* operation exports the selected components from the ACE environment to the import/export directory. Selecting *Utility* > *Export* opens an Application tree dialog box which enables you to select the components to export. The tree contains only the components that exist in the local ACE environment.

Example 4. Export operation

This is the *Export Application Components* dialog box for an Application named EXAMPLE, which has two libraries - EXLIB1 and EXLIB2.

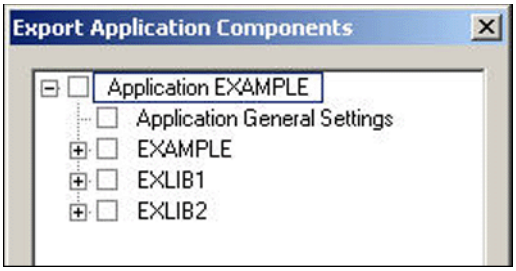


Figure 6. Export Application Components dialog box

When invoking the *Export* operation, the following occurs:

- If a Subapplication is open in ACE, it is automatically closed.
- If an Application is open in ACE, this Application is displayed in the Application tree dialog box.
- If no Application is open in ACE, you first select an Application, and then the selected Application is displayed in the Application tree dialog box.

> **Note:** When no Applications exist in ACE, the *Export* option is disabled.

During the Export operation, which can be lengthy, a progress dialog box is displayed. This dialog box enables you to terminate the operation.

Import Operation

The *Import* operation imports the selected components from the import/export directory into ACE. Selecting *Utility* > *Import* opens an Application tree dialog box which enables you to select the components to import. The tree contains only the components that exist in the import/export directory.

Example 5. Import Operation

This is the *Import Application Components* dialog box for an Application named EXAMPLE, which exists in the import/export directory but without any libraries:

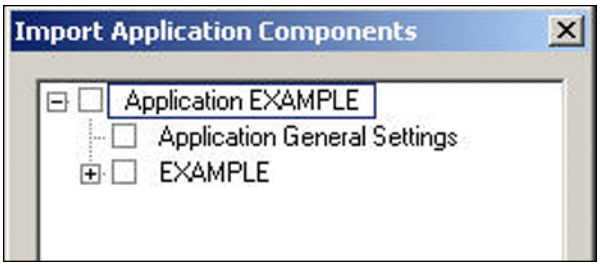


Figure 7. Import Application Components dialog box

When invoking the *Import* operation, the following occurs:

- If a Subapplication is open in ACE, it is automatically closed.
- If an Application is open in ACE, the Application tree dialog box displays the Application components that exist in the import/export directory.

  When the import/export directory does not contain any components which belong to the open Application, a warning message is displayed to the ACE developer.
- If no Application is open in ACE, you first select an Application from a list of Applications which have components in the import/export location. Then, the

Application tree dialog box displays the components of the selected Application.

- When the import/export directory does not contain any Applications, a warning message is displayed.

Importing Application Components

When several ACE developers work on an Application together, it may be necessary to transfer parts of the Application from one developer to another.

Due to the hierarchical structure of an ACE Application, there are dependencies between an Application's components. Therefore, the order in which you import components into ACE is important. For example, you cannot import a Subapplication if its library does not already exist in ACE, and you cannot import a library if its Application does not already exist in ACE. In addition, there are dependencies between the components which exist in the import/export directory and those which exist in the ACE environment to which you are importing files. The components in the *Import Application Components* dialog box are automatically enabled or disabled according to these dependencies. For example, some components might be disabled in the *Import Application Components* dialog box in spite of the fact that they do exist in the import/export directory.

Rules to Remember

The following rules apply when working with the *Import Application Components* dialog box:

- The entire Application tree is disabled:
  - When the Application does not exist in the local ACE environment, and the Application general settings component does not exist in the import/export directory.
  - When the Application does not exist in the local ACE environment, the Application general settings component does exist in the import/export directory, but there is no child node with the name of the Application in the import/export directory.
  - When the Application does not exist in the local ACE environment, the Application general settings component does exist in the import/export directory, there is a child node with the name of the Application but this child node does not contain a general settings node in the import/export directory.
- A library and its child nodes are disabled when the library does not exist in the local ACE environment, and the library general settings component does not exist in the import/export directory.

- When the Application does not exist in the local ACE environment, selecting the Application general settings node in the root Application automatically selects the library general settings node of the Application as well.
- When the Application does not exist in the local ACE environment, the Application general settings node is selected and disabled if at least one of its siblings is selected. If no siblings are selected, the Application general settings node is not disabled.
- When a library does not exist in the local ACE environment, the library general settings node is selected and disabled if the Subapplications node is selected. If the Subapplications node is not selected, the library general settings node is not disabled.

Handling Principal and Dependent Subapplications

There are three types of Subapplications: regular, principal and dependent. The principal and dependent Subapplications are used in the Many-to-One feature, and are handled differently than regular Subapplications.

In order to easily differentiate these Subapplications from regular Subapplications, their package file names are:

```
<SubApplName>.principal.jps
```

```
<SubApplName>.dependent.jps
```

In the Application tree dialog box, a principal Subapplication is presented as a parent node, and its dependent Subapplications are its child nodes. However, a principal Subapplication and its dependent Subapplications are considered one unit. Therefore, it is not possible to select a dependent Subapplication, and selecting or clearing the principal Subapplication automatically selects or clears all of its dependent Subapplications.

> **Note:** A dependent Subapplication for which no principal Subapplication has yet been defined, is presented in the tree as a regular Subapplication.

For more information on the Many-to-One feature, see Chapter 10 - "Many-to-One" on page 147.

Considerations When Using Configuration Management

The following assumptions need to be made when using configuration management in an ACE project:

- Only one ACE developer can modify a specific Subapplication concurrently. This also includes the *.DDO file that was used to create the Subapplication.

Since one *.DDO file can produce more than one Subapplication, the same ACE developer must handle all Subapplications created from this DDO file.

- Only one ACE developer can modify global general settings concurrently. These global settings refer mainly to KnowledgeBase files and methods files.

Chapter 2. Packaging Applications

Packed files, or packages, are “archives” used for distributing and storing ACE Applications. A package contains one or more files, compressed to save space. Applications can be packaged in ACE on one computer and then unpacked on other computers. Packaging Applications saves time and space, and makes transferring e-mail attachments faster.

Consider packaging an Application when:

- Several developers are working on different parts of the same Application in a multiple-developer environment.
- Sending part, or all, of an Application to customer support.
- Creating archives for backup purposes.

This chapter describes:

- Packaging Applications

Packaging Applications

The need for Application/library transfer generally arises when the database files on the machine of a library developer require additions or fine tuning. The library developer then transfers the library to the central developer. The central developer modifies the Application-level elements as needed, and transfers the updated development environment to the library developer.

This ensures consistent updating of the development environment and efficient work assignment amongst developers.

For example, when one central developer is working on Application-level elements (such as Methods, formatting dictionaries, *.ini files and KnowledgeBases) and other developers are working with other elements (such as libraries) belonging to the Application, the main use of Pack/Unpack is easy transfer of Applications and libraries between the central developer and the other developers.

Note: JIS 9.3 uses SQLite for application and library data. Unpacking eligible BDE packages starts migration automatically. Private BDE components are supplied only for conversion; no system-wide BDE installation is required. Retain backups until validation is complete.

Contents of a Package

You can choose to pack an Application, a library, or a subset of Subapplications from an Application or library.

The package can contain:

- For each Application or library, all the files in its directory unless only some of the Subapplications were selected.
- All KnowledgeBase files used by the Application or library, even those located outside the Application or library directory.
- All *.ini files used by the Application or library. This also includes the relevant database information.

You can also include other files that are related to the Application or library in the package. These include:

- Visual Basic, HTML, and/or Java files.
- Bitmaps, even when located outside the Application or library directory.
- Files in the input directories: DDS, SDF, and screens.
- Files in the output directories: Temporary Generate Runtime, Runtime, and Runtime Installation.
- Other files on your computer.

The Packing Process

To pack an Application, a library, or some part of either:

1. From the *Utility* menu, select *Pack*. The *Pack* wizard opens.
2. If you start the wizard before you open an Application, the first step asks you to specify the name of the Application to pack.

   If you start the wizard from within an open Application, the wizard skips this first step and goes on with the packing process for the open Application.
3. Specify a name and a storage location for the packed file.
4. Follow the instructions in the wizard until the last step. By following the steps in the wizard you:
   - Select which content to pack, including the Application itself, and possibly some specific Subapplications from each library or Application.
   - Specify whether to pack as a single file or as several files. When the package is divided into a number of files, one file receives the extension *.JPK, the other files receive the extensions 000, 001,002 and so on.
   - Can choose to include other files in the package. This includes external files, such as Application-level files, bitmap files, or other additional files of your choice.

Note: If several bitmaps or additional files are added to the package, make sure to enter each file's name in quotes when manually entering their name in the wizard.

5 When you click *Finish*, the packing process takes place and a JIS Package file is created. A JIS Package file has a *.JPK extension.

Note: When an Application or library is manually copied, the appls.ini file does not contain a reference to its specific.ini file. In this case, the Application or library cannot be packed. Attempting to pack Applications or libraries that do not have their specific.ini file listed in ACE's appls.ini, generates an error message in the *Pack* wizard. To successfully pack the Application or library: Exit ACE and add the appropriate reference in the ACE\kb_400\APPLS.ini (for an iSeries Application), or the ACE\kb_3270\APPLS.ini (for a mainframe Application).

The Unpacking Process

Like the packing process, the unpacking process is wizard-driven. The *Unpack* wizard lets you select a JIS Package file and unpack it. You can choose to unpack part of the package.

After all the relevant information is gathered by the wizard, the unpacking process takes place.

Note: Before you unpack a full or partial application or library, determine if you have a backup of the application or library that you are updating or replacing. If you do not have a backup, consider whether you want to create a backup first, before proceeding with the unpack. You could actually use the *Pack* function to create a backup, provided that you do not overwrite the same `.jpk` file that you want to restore!

To unpack an application, a library, or some part of either:

1 From the *Utility* menu, select *Unpack*

2 In the first step - *Specify Package to Unpack* - enter the name of the package to be unpacked. If the package is divided into a number of files, enter the name of the file with the *.JPK extension.

3 In the step called *Specify Content to Unpack*, specify the libraries and Subapplications to be unpacked.

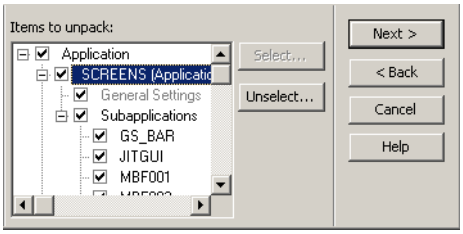


Figure 8. Specifying the libraries and subapplications to unpack

Libraries and Subapplications are selected for unpacking, or deselected, by setting or clearing the check boxes next to the files, or by clicking the *Select* or *Unselect* buttons.

When the *Select* or *Unselect* button is clicked, the *Add/Remove subapplications from selection* dialog box is displayed.

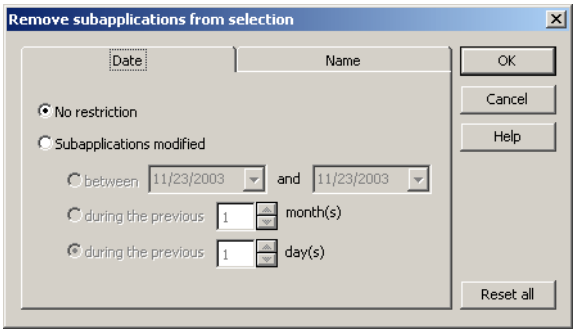


Figure 9. Add/Remove subapplications from selection dialog box

Use this dialog box to add or remove Subapplications according to two types of search criteria:

- By modification date
- By name

Search is conducted through both tabs. Leave the default **No Restriction** selected when selecting Subapplications that are not based upon the last modification date.

To select Subapplications without name restrictions, in the **Name** tab, leave the edit fields blank.

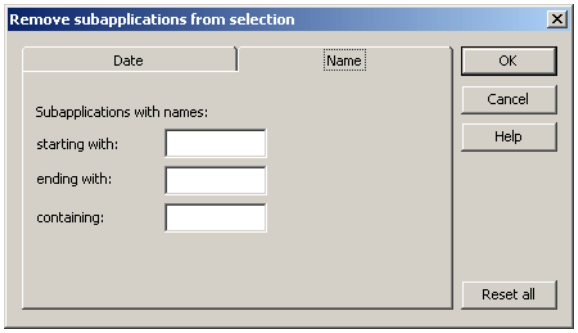


Figure 10. How to select subapplications without name restrictions

Data in this dialog box is saved in an *.ini file for subsequent use. Click *Reset all* to clear search criteria and to display default settings for both tabs.

When *Select* is clicked, all unselected Subapplications that match the search criteria are added to the already selected Subapplication.

When *Unselect* is clicked, all selected Subapplications that match the search criteria are removed from the already selected Subapplication.

Search criteria can be run several times on existing selections of Subapplications, thus narrowing or expanding the selection each time the criteria is run.

4 The *Specify Target Application Name* step gives the option of changing the name of the Application while unpacking.

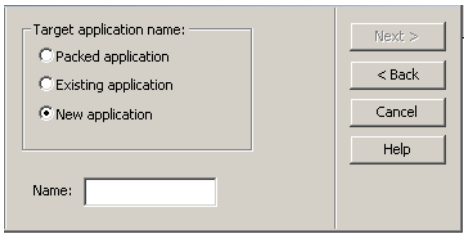


Figure 11. Specify Target Application step

Choose from the following three options:

- *Packed application* - This unpacks the Application under the same name.
- *Existing application* - This overwrites an existing Application with the Application being unpacked. Choosing this option opens a combo box containing the names of the existing Applications.
- *New application* - Unpack the Application under a new name. Choosing this option opens an edit box where you type the new name.

Note: When assigning a new name to the Application, a warning message is generated. In some types of files, such as Java extensions, the Application reference is not automatically updated to the new name. These instances must be updated manually.

5 The *Unpack* wizard gives you the option to change the name of a library while unpacking it. If you chose to unpack libraries, you can change their name in the *Specify Library Name* step.

6 Unpacking an Application and/or libraries to an existing Application or set of libraries, overwrites the existing information. The *Update or Replace Existing Applications/Libraries* step has 2 options:

Update - Retains information that is not overwritten when unpacking.

Replace - Removes all information that is not overwritten during unpacking.

7 In the *Specify Additional Files to Unpack* step, a list of directories, and additional files which were included in the package, are displayed in a tree. This step is automatically skipped, if no such files were added to your package.

Note: The list does not display a typical directory tree. If files are added from the `c:\appl\files` directory, and then files are added from the `c:\appl\files\mbf` directory, the *Unpack* wizard displays two separate trees instead of one tree with subfolders.

The tree consists of rows of cells. Directories and file names are listed alphabetically. When several directories with the same name (but with different file lists) exist in the package, they are concatenated into one directory node. Expand the tree to see the list of files in each directory.

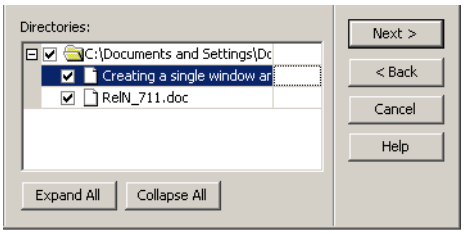


Figure 12. Directory and file tree

Files can be selected for unpacking, or deselected, by setting or clearing the check boxes next to the files.

Specify a new destination directory where the files will be unpacked by selecting the appropriate directory cell and typing the new directory name.

8 Click *Next* to move on to the next step in the wizard. Follow the instructions in the wizard until the last step. The next few steps of the wizard are for optionally unpacking extra files included in the package. See “Contents of a Package” on page 40.

9 When you click *Finish*, the unpacking process takes place, and all relevant files are extracted from the package to their appropriate locations in the ACE directories.

10 In the process of unpacking, if you chose “Update Existing Application Libraries” (instead of “Replace”) you may be shown a message box like this:

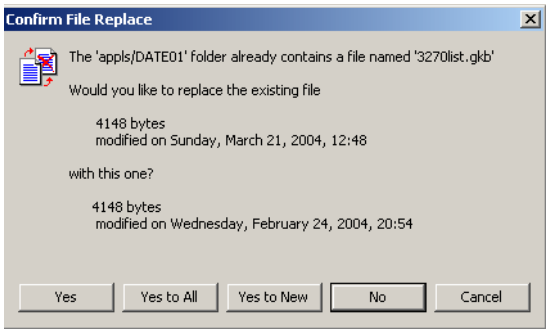


Figure 13. Confirm File Replace message box

This message box indicates that the wizard found a file on the .jpk that is older than an identically named file in the target application or library.You are given several alternatives:

- *Yes* - restore the file in question.
- *Yes to all* - restore the file in question and all subsequent files, even if they are older than the files being overwritten.
- *Yes to new* - restore only newer files; do not restore a file if it is older than the file it would overwrite.
- *No* - do not restore the file in question.
- *Cancel* - cancel the unpack operation.

Select the appropriate alternative.

11 Run Generate Runtime on the unpacked libraries and Applications.

Important: If unpacking a new library to an Application, perform Generate Runtime on the Application and on the library.

It is recommended to run the Generate Runtime process on all Subapplications and not only those modified since the last Generate Runtime.

Unpacking KnowledgeBase Files

KnowledgeBase files, files with a *.GKB extension, may be stored on multiple drives. The location of KnowledgeBase files used by an Application is listed in the specific.ini file of that Application. The Pack utility packs the general KnowledgeBase files accordingly. When running Unpack, ACE initially attempts to unpack the GKB (General KnowledgeBase) files to the exact location where they were found when the Application was packed. If the path is not found (e.g. a network drive does not exist when unpacking), the utility prompts for an alternative unpack location.

Note: The specific.ini file must be edited when there is a change in the location of the GKB files.

Unpacking a Library without the Main Application

If you want to unpack a library, without unpacking the main Application, to an environment that only contains the main Application, or that contains the Application and another library, you must add the library name to the Application's library list. This must be done before unpacking the library.

To add the library name to the list:

1. Make sure the server is running and open the JIS Administrator.
2. Go to the *Runtime Configuration* tab and from the left pane Application tree select the Application that the library is being unpacked to.
3. From the *Category* list, select *Navigation*. Enter the necessary changes in the *Application libraries* field.

Note: If your runtime is set to Autostart, change the <ApplName>.ini setting manually. Add the unpacked library to:

```
[Program]

Libraries=
```

Packing and Unpacking KnowledgeBase Files

The Packing/UnPacking feature allows storing *.gkb files on multiple drives. The location of KnowledgeBase files (*.gkb) used by an Application is listed in the specific.ini file of the Application. When running Unpack, ACE initially attempts to unpack the *.gkb files to the exact location where they were found when the Application was packed. If the path is not found, the utility prompts for an alternative unpack location.

Attention: The specific.ini file must be edited when there is a change in the location of the *.gkb files.

Miscellaneous Remarks

- Pack and Unpack operations can be interrupted by the user at any time.
- In the Specify Screen Images Directories to Pack step of the Pack wizard, the DDS, SDF, and Screens check boxes are checked by default regardless of the nature of the Application—iSeries or mainframe. Checking all the options is

unnecessary because the DDS directory does not exist in mainframe applications and the SDF directory is empty in iSeries applications.

- In the Specify Output Directories to Pack step of the Pack wizard, the Temporary Auxiliary, Runtime, and Installation check boxes are checked by default, even when you have not run Generate Runtime or produced a Runtime Installation yet. When this is the case, these directories may not exist or may be empty.

Limitations

- When running under Windows NT, the focus is not automatically returned to the wizard after an error message box is displayed. Use a mouse click or the Alt+TAB keys to return to the wizard.
- When switching to another application while the Pack or Unpack wizard is open, you must return to ACE using the Alt+TAB keys. If you try to return by clicking the icon in the task bar, the main window is disabled and the wizard does not show.

Part II. About Advanced Use of Design

This section explains two ACE features designed to control the way certain items appear on the screen and how they are sent back to the host. Date Control gives you the ability to configure the date presentation on the GUI in a number of different formats. This comes in handy when host applications containing date information from one country are being viewed in another country. Often the date format differs between countries and might cause confusion at the user level. ACE provides a way to make this situation work for you.

User Formatting Functions are used to give text a uniform appearance throughout the Application. By identifying the text according to Pattern Definitions that define them, you can configure that text to appear in customized ways. For example, text can be displayed in all caps, begin with a capital letter or include other characters not inherent in the host string.

Chapter 3. Date Control

Creating a Date control is accomplished though Design View. The control displays the date and has an associated calendar. There are several ways to change the calendar date by using either the keyboard or the mouse. ACE also allows you to change the way the date is formatted.

Formatting the way in which ACE writes the date on the host can be done using a GUI interface or directly from an *.ini file. Formatting the date via the *.ini file has several advantages. Both methods of date formatting are discussed in this chapter.

This chapter describes:

- Creating a Date Control
- Formatting the Date Presentation in the GUI Window

Creating a Date Control

A Date control gives you the ability to configure the date presentation on the GUI in a number of different formats. This comes in handy when host applications containing date information from one country are being viewed in another country. Often the date differs between countries and might cause confusion at the user level. ACE provides a way to make this situation work for you.

The ACE date control is a specialized edit field that enables your users to easily enter data that is intended to be interpreted as date information. The date control contains the following features:

- Format masking that limits the allowed input characters.
- Year data input as two digits is automatically translated to four digit data using the defined base year.
- A calendar tool associated with the date control allows users to choose a date graphically

ACE tracks the date using the date settings according to the *Windows* operating system. You can change the *Windows* date settings by accessing the *Regional Options* dialog box located in the *Windows Control Panel.* ACE also provides several ways to control how the date is sent to the host. For an even higher level of customizing you can edit the ACE *.ini files to configure special date items.

To insert a Date field using the *Definitions Palette*:

1 Go to Design View.

2 In the *Definitions Palette*, select *Date* and drag it to the desired location on the window.
3 Resize using the resizing handles. Change location using drag and drop.
4 To test the new control, go to Test View and click on the arrow to open the calendar.

To insert a Date field through the Design menu:

1 In Design View, from the *Design* menu select *Add Control*.
The *Add Control* dialog box appears.

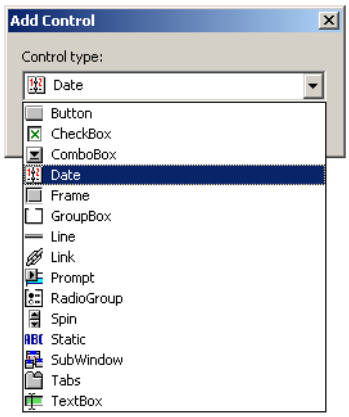


Figure 14. Add Control dialog box

2 Select *Date* and click *OK*. The *Date Component* dialog box appears.
3 Leave the *Date Component* fields at their default states and click *OK*. A date field is added in the upper left corner of the Subapplication window.
4 Resize using the resizing handles. Change location using drag and drop.
5 To test the new control, go to Test View and click on the arrow to open the calendar.

Note: The date field is blank until it has been tested in Test View. After testing, the date field shows the current date in the selected format. Pressing the arrow displays a monthly calendar.

Using the Calendar

The date displayed in the Date box has 3 fields: day, month, and year. The order of the fields depends on the date format settings.

The date field has an associated calendar. The calendar shows an entire month. Clicking on any date in the calendar registers the selected date inside the date field. The date entered in the date field is received by the host.

Opening the Calendar

To open the calendar in Test View do any one of the following:

- Place the cursor in the date field, and press *F4*.
- Place the cursor in the date field, and press *ALT* + *Ø* on the keyboard.
- Click the arrow button adjacent to the date window.

Figure 15. Calendar control

Closing the Calendar

To close the calendar, press *Escape*, *Enter*, or *click on the arrow button*.

Navigating the Calendar

- Within the date field the left/right arrow keys jump from day to month to year.
- The up/down arrow keys advance the number in the field (day, month or year).
- When changing the year:
  Going forward – Jump from December to January of the next year.
  Going backward – Jump from January to December of the previous year.
- When changing the month or the year, the day remains constant.

You can change the date in any of the following ways:

Table 5. Changing the date (Sheet 1 of 2)

	In Calendar (calendar must be open)	In Date Fields
Date	Click on a date. The calendar closes. The selected date is displayed in the date field.	Place the insertion point in the day field. Use the up/down arrow keys on the keyboard to advance or go back.

Table 5. Changing the date (Sheet 2 of 2)

	In Calendar (calendar must be open)	In Date Fields
		Place the insertion point in front of the digit and type a new digit. **Example:** Change 23 to 28. Place the cursor between the 2 and the 3 and type 8.
Month	Use the arrows on the calendar.	Place the insertion point in the month field. Use the up/down arrow keys on the keyboard to advance or go back.
	Use the Page Up and Page Down keys on the keyboard. Page Up – Jump to previous month Page Down – Jump to next month.	Place the insertion point in front of the digit and type a new digit.
Year	Use the arrows on the calendar (see figure)	Place the insertion point in the year field. Use the up/down arrow keys on the keyboard to advance or go back.
		Place the insertion point in front of the digit and type new digit.

Changing the Short Date Format

The way ACE displays the date is determined by the *Windows* user settings. The user can control these settings using the *International Dialog*.

To change the Short date Format:

1. From the *Start* menu, select *Control Panel*
2. From the *Control Panel*, select *Regional Options*.

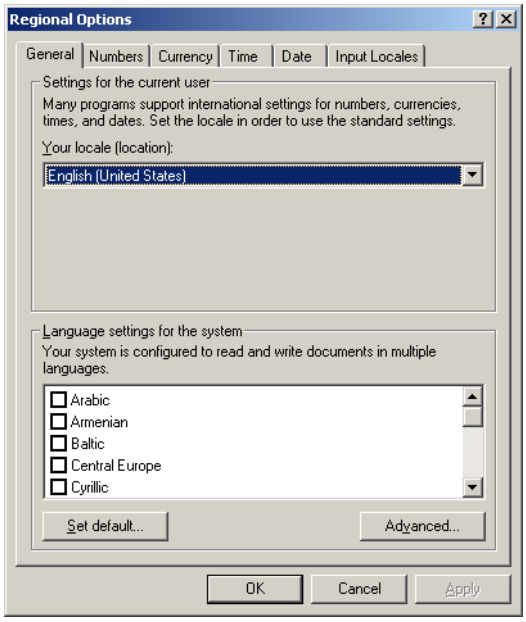


Figure 16. Regional Options dialog box

3 From the *General* tab choose your locale.

4 Switch to the *Time* tab and enter the required information in the fields provided. Repeat Step 4 for *Date* tab.

5 Click *OK* to save your time and date configurations in *Windows*.

Note: The date format that is chosen here also appears in the same format in the date control that is added to your ACE Application. Read the Microsoft Windows manual for instructions and explanations concerning the *Regional Options* dialog box.

Language Localization

Strings and date conventions in the date control are also determined by *Windows* regional settings, with the following limitations:

- Languages are supported in 32-bit runtimes.
- English, French, German, Spanish and Swedish are also supported in 16-bit runtimes.

Formatting the Date Presentation in the GUI Window

This option is used to define the way ACE analyzes the date that appears on the host and the format in which ACE writes the date to the host. To help ACE perform these operations, use the format functions. Two functions are added to the *Text Format Definition* dialog box:

For Window to Manager	FormatACEDate
	FormatACEDateFromIni
For Manager to Window	FormatHostDateFromIni

FormatHostDate and FormatACEDate

To format the host date and ACE date first create a Date control component. See the first pages of this chapter for instructions on how to create Date controls.

To format host date and ACE date information:

1. In Design View, double click the date control on the GUI window. The *Date Component* dialog box appears.
2. In the *Format* tab, click the *Format* button. The *Text Format Definition* dialog box appears. In this dialog box, specify *FormatHostDate* when it appears as the output and/or *FormatACEDate*, when ACE updates the data to the Host.

   The *Text Format Definition* dialog box contains two list boxes. The left list box, entitled *Available*, lists all the available formatting options. The right list box, entitled *Selected*, lists the selections that the user has chosen.
3. From the *Available* list, select *FormatHostDate* (for Manager to Window), and press *Add*. The option is added to the *Selected* list box.
4. Select *FormatHostDate* from the *Selected* list box, and press *Parameters*. The *Format Date* dialog box appears.

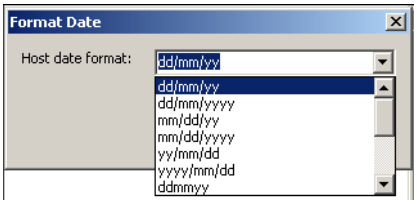


Figure 17. FormatDate dialog box

5. In the *Host date format* combo box, specify a parameter that describes the date on the host screen, and press *OK*. The user can select a format string from the combo box or write a new format string.

The same set of steps should also be carried out for *FormatACEDate* (for Window to Manager).

Make sure that the string is valid in such forms as: dd/mm/yy or dd/mm/yyyy:

- The d remains next to the d; dd represents the date in 2-digit format.
- The m remains next to the m; mm represents the numerical equivalent of the month. March is represented as 03. mmm represents the 3 letter abbreviation of the month. Jan, Feb, Mar, etc.
- The y remains next to the y; y represents the year in either two digit or four digit format.

The Advantages of Using FormatHostDateFromIni and FormatACEDateFromIni

Since function parameters are written to the FORMATS.ini file, you can change a format and its values without the need of recompiling the entire Application.

It is possible to assign more than one value to a date format.

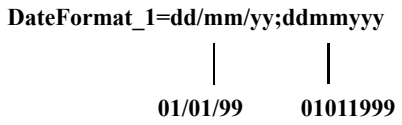


Figure 18. Assigning more than one value to a date control

ACE scans the date format from left to right to find the format string that matches the date pattern. When formatting can be performed, scanning stops. To save time, format the parameter so that the format string that is most likely to match the date pattern on the host is placed to the left of the parameter. Other format strings representing date patterns that appear less frequently should be placed to its right.

A function can have more than one parameter. You can format each of the parameter's values to accommodate date settings of different countries.

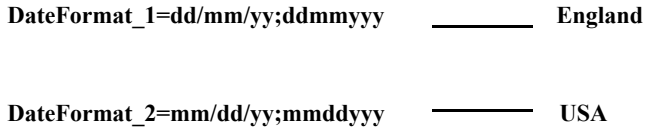


Figure 19. Formatting more than one parameter

This means you can use a Date representation as a generic date setting and change the parameters values according to countries' specific date patterns.

FormatHostDateFromIni and FormatACEDateFromIni

These two functions allow for more precision when specifying the Date format. They also enable you to modify the Date format in the runtime Application, via the *.ini file. To format date information using an *.ini file, first create a Date control component. See the beginning of this chapter for instructions on how to create Date controls.

To format host date and ACE date information from the *.ini file:

1. In Design view, double click the date control on the GUI window. The *Date Component* dialog box appears.
2. In the *Format* tab click the *Format* button. The *Text Format Definition* dialog box appears. In this dialog box, specify the *FormatHostDateFromIni* when it appears as the output and/or the *FormatACEDateFromIni*, when ACE updates the data to the host.

   The *Text Format Definition* dialog box contains two list boxes. The left list box, entitled *Available*, lists all the available formatting options. The right list box, entitled *Selected*, contains all the selections the user has chosen.
3. From the *Available* list, select *FormatHostDateFromIni* (for Manager to Window), and press *Add*. The option is added to the *Selected* list box.
4. Select *FormatHostDateFromIni* in the *Selected* list box, and click *Parameters*. The *Format Date Parameter* dialog box appears.

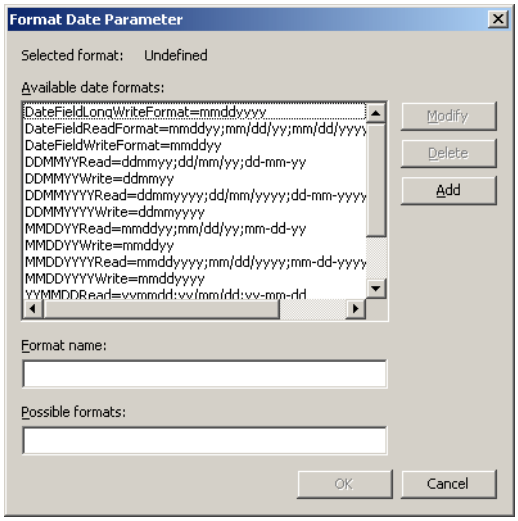


Figure 20. Format Date Parameter dialog box

   In this dialog box you format the date parameter that represents the date on the host screen.
5. In the *Possible Formats* field configure the format value strings. These strings depict the different ways a date can be represented. You can enter a list of several date format strings. Separate format strings with semicolons.

6 In the *Format Name* field enter a meaningful name. This symbolic name represents the date format string or strings. Click the *Add* button. The new format name appears in the *Available Date Formats* list.

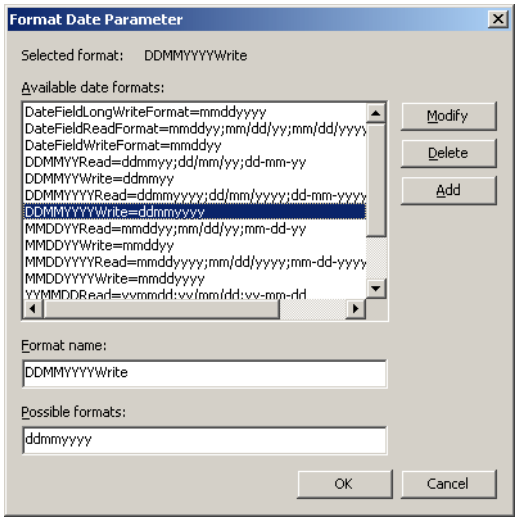


Figure 21. Available Date Formats list

The selected parameter will define the *FormatHostDateFromIni* function. Press *OK* to attach it to the function.

7 The same set of steps should also be carried out for *FormatACEDateFromIni* (for Window to Manager).

When you compile an Application, the formats' names and their different values are written to the `FORMATS.ini` file in the following form:

```
[DateFormat]
DateFormat_1=dd/mm/yyyy;dd/mm/yy;ddmmyy;ddmmyyyy
MonthsAbbreviations=Jan;Feb…….Dec;
```

Note: `FORMATS.ini` files are located both in the converter and the runtime directories. Make sure you copy the converter version of the file into the runtime directory after the definition is completed.

Hosts that Change Date Information On-The-Fly

Some host applications require more than one format for date representation. For example, the end user's host may communicate between two sites; one in North America and one in Europe. In this case, a screen of a host application may require the date March 16, 1998 to display on the end user's PC in North American format; the month appearing first followed by the day and then the year. When the data from the PC is sent to the user in Europe the date must be in European format; the day appearing first followed by the month and then the year.

ACE allows you to configure an Application so that the date format changes on-the-fly. This is accomplished by editing the formats.ini file and using the *SwitchDateFormats* DoMethod. During runtime the DoMethod automatically changes date formats which are written to the formats.ini.

Example 6. Hosts that change date information on-the-fly

Using the scenario above you would enter the following lines to your FORMATS.ini file with a text editor.

```
[DateFormat]
USA_R=mmddyy;mm/dd/yyyy
EUR_R=ddmmyy;dd/mm/yyyy
DateReadFormat=...To be determined by the host application.
```

Note: The USA_R and EUR_R fields represent the date field structure of the date format for this particular example. For your Application enter other field names instead of USA_R and EUR_R.

`DateReadFormat` is the parameter that is used in the formatting function in the Representation Definition of the control. It is also the current format being used. If you want to switch formats from either USA to EUR or, EUR to USA you need to copy the contents of either EUR_R or USA_R to the *DateReadFormat* keyword. This is accomplished by the DoMethod *SwitchDateFormats*. The receiver of this method is the Application.

Creating the SwitchDateFormats DoMethod

Switch to Design View to create the *SwitchDateFormats* DoMethod then:

1. Create a Date control. See the beginning of this chapter for instructions on how to create Date controls.
2. Select *Design* > *User-Triggered Methods*.

   The *User-Triggered Methods* dialog box opens.
3. Select *New* > *Current Subapplication Method*. The *Define Method* dialog box appears.
4. Give the new method a logical name and double-click *DoMethod* in the *Line type* list box. The *DoMethod: Method Activation* dialog box appears.
5. Select the *SwitchDateFormats* DoMethod and click *Assign Values*. The *Method Parameters* dialog box appears.
6. In the *Parameter Name* box choose either the current date format or a new format.

7 In the *Parameter Value* field enter the ***DateReadFormat***.

DoMethod:Receiver: Text is entered as the following including the quotation marks. Your entry may differ from the one below depending on your applications requirements but the format will be the same.

"mmddyy:mm/dd/yyy" Click *OK* twice. You are left with the **Define Method** dialog box still open. In the lines pane check that the method line is written accordingly.

```
'Application'Method:SwitchDateFormat
Parms: ("DateReadFormat, "USA_R")
```

Click *OK* to close the *Define Method* dialog box and save the new Method.

DateBaseYear

The GUI can handle dates spanning a 100 year interval. By default, this interval is 1940 to 2039. The low end of the interval is called the DateBaseYear.

To change the Application's date base year:

1 Use a text editor to open <InstallDir>\APPLS\RT32\<ApplName>.ini

2 In the `[Program]` section edit the value of the `DateBaseYear` key. The value you choose is the new low end of the 100 year interval supported by the GUI.

Chapter 4. User Formatting Functions

User Formatting Functions are used to give text which appears throughout an application a uniform style. Text derived from the same Pattern Definition can be formatted by creating a DLL that can be called up during conversion. In addition, the user can create an application-specific dictionary to redefine terms commonly used in an application.

This chapter describes:

- Changes in the Converter.ini File
- Identifying Pattern Definitions
- Using the Dictionary

Editing the converter.ini file is the first step when applying User Formatting Functions to your application.

Next, you select the Pattern Definitions which will receive the automatic formatting.

The last topics of this chapter show you how to create dictionaries containing definitions which can then be applied to your entire application.

Changes in the Converter.ini File

Before using this feature, it is necessary to edit the converter.ini file (cnvrt400.ini for iSeries applications, and convert.ini for mainframe applications). Verify that the DLL you are interested in using is represented and flagged in the converter.ini file. To do so, the following lines should be added to the converter.ini file:

```
[UserDefinedFormat]
UseUserFormat=1
Dlls=frmfun32.dll
```

In the example above we are using the frmfun32.dll. If other formatting DLLs are desired, and available, they can be included by adding a semicolon and the DLL name. The format would be as follows:

```
Dlls = frmfun32.dll; format2.dll; format3.dll;
```

When this is added to the converter.ini file, you can use this feature to format the characteristics of a text-related Pattern Definition. The following is the procedure for identifying a Pattern Definition, and formatting it using the dictionaries.

Identifying Pattern Definitions

In this example we will use the *Header* Pattern Definition.

1. In Analysis View, click on the field which you would like to format.
2. Press *F5* to scroll through each Pattern Definition associated with that field. The names of the Pattern Definitions appear under *Selected pattern* in the *Pattern Definition Properties* dialog box.

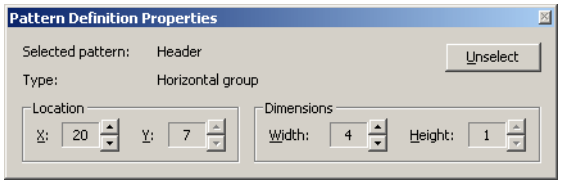


Figure 22. Pattern Definition Properties dialog box

3. When the Pattern Definition is identified, press *F8* to open the KnowledgeBase at the selected Pattern Definition.
4. From the Tool bar click the *Representation Definition View* icon .
5. In Representation Definition View, open the *Format* tab.
6. Click the *Format* button. The *Text Format Definition* dialog box appears.

   If the selected component is not associated with text, this button is not active.

If you did not initially edit the *.ini file, ACE does not allow access to this function; in this case, the following error message appears:

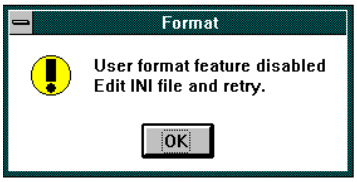


Figure 23. User format feature disabled warning message

Text Format Definition Dialog Box

The Text Format Definition dialog box contains two list boxes. The left list box, entitled **Available,** lists all the available formatting options. The right list box, entitled **Selected,** contains all the selections the user has chosen.

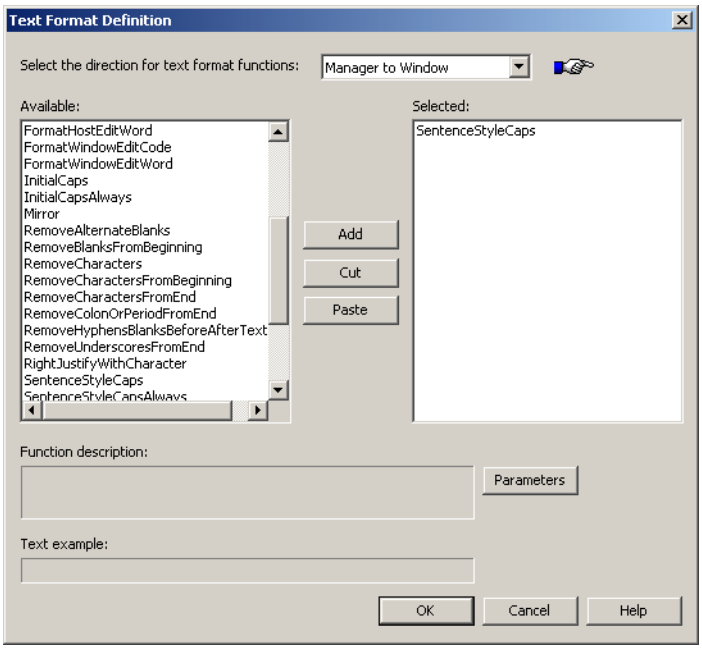


Figure 24. Available and Selected lists in the Text Format Definition dialog box

Three buttons are located between the list boxes, for transferring the options to and from the list boxes.

The three buttons are as follows:

Table 6. Text Format Definition dialog box buttons

Button	Description
Add	Copies the formatting options you have selected in the *Available* list box and places them in the *Selected* list box.
Cut	Removes selection(s) highlighted in the *Selected* list box.
Paste	Pastes the edited selection(s) in a new location within the *Selected* list box.

Beneath the list boxes are two additional fields:

Table 7. Text Format Definition dialog box fields

Field	Description
Function Description	Describes what the highlighted function does.
Text Example	Provides an example of how your selection will appear in your Application.

Using the Text Format Definition Dialog Box

Highlight your selection in the *Available* list and press the *Add* button to add it to the *Selected* list. The order of your selections is reflected in the text. Therefore, if for example, you want your formatted text to include a colon followed by three periods, first select *AddColonAtEnd* followed by *AddThreePeriods*.

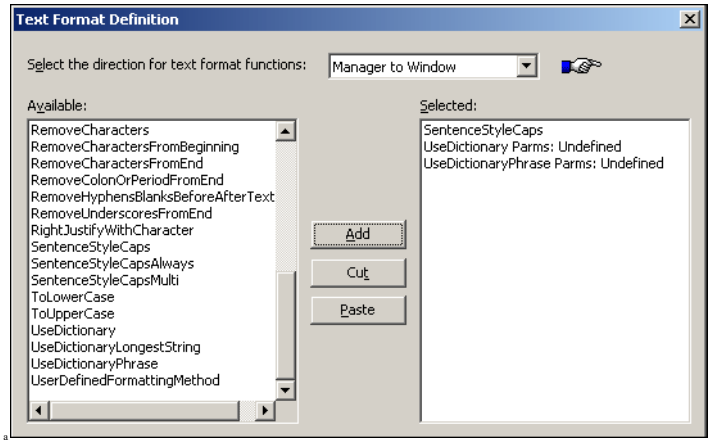
Using the Dictionary

ACE enables you to create various dictionaries containing definitions which can be applied to your entire Application. The dictionary options are accessed through the list of *Available* options in the *Text Format Definitions* dialog box, and are entitled *UseDictionary* and *UseDictionaryPhrase*.

Table 8. Dictionary options

Dictionary	Description
UseDictionary	This is a single-word dictionary. It considers each word as separate text. Enter the word to translate, and the value to be substituted.
UseDictionaryPhrase	This is a dictionary of text strings. Enter the string to be translated and the value (string) to be substituted.

To use these dictionaries you must first add them to the *Selected* list. Click on either of these options in the *Available* list box and press the *Add* button. The dictionary type selected then appears in the *Selected* list box.



When using the dictionary options, the dictionary entries should appear first in the list of selected options. Otherwise, the assigned word may be altered by another command and will not be recognized by the dictionary. To rearrange options within the *Selected* list, use the *Cut* and *Paste* buttons.

The string `Parms: Unidentified` is displayed with the selection. This indicates that a dictionary has not yet been selected. Double click the entry to create or select a dictionary.

The following example illustrates the difference between the two dictionaries:

```
Pls enter order no.
```

The *UseDictionary* selection reads each word individually and makes adjustments accordingly. If the definitions `Pls=Please` and `no.=number` were defined within your *UseDictionary* then this phrase would correctly read `Please enter order number.`

UseDictionaryPhrase considers these words as a string, searches for matching definitions within its dictionary, and replaces it with a string. For example, if new order numbers were issued and you wanted the new order numbers to be entered within the application, your definition may read as follows:

```
Pls. enter order no.= Please enter NEW order number
```

Thereby, the original phrase would be replaced with `Please enter NEW order number.`

Calling up a Dictionary

Call up a dictionary by either double clicking one of the dictionary types in the *Selected* list box, or by clicking the *Parameters* button. The *List of Dictionaries* dialog box opens.

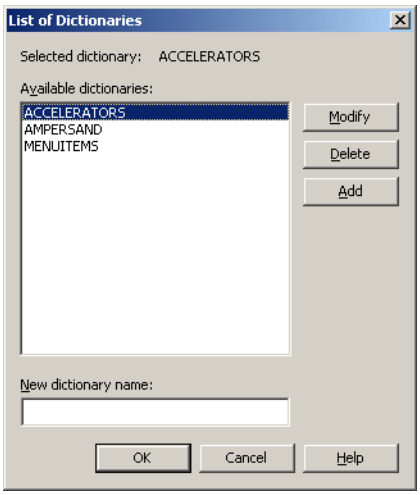


Figure 25. List of Dictionaries dialog box

If dictionaries have been defined, they are displayed in the *Available dictionaries* list. Select one of the dictionaries from the list.

Creating a New Dictionary

If a dictionary has not yet been defined, you can create a new dictionary. To do so, follow these steps:

1. In the *New Dictionary* field, enter a new dictionary title (e.g. ACRONYMS) and click *Add* to add this name to the list of available dictionaries.

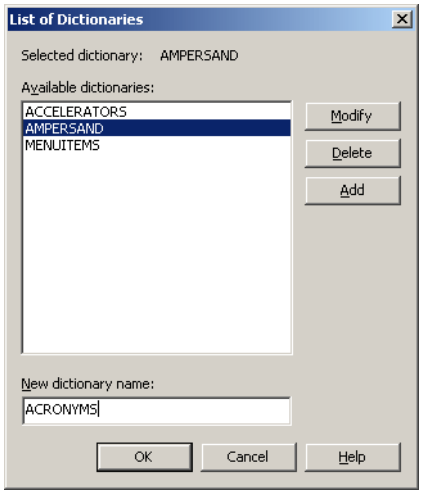


Figure 26. List of Dictionaries dialog box - New Dictionary field

2 Click *OK*. The new dictionary is added to the *Available dictionaries* list.

Entering Definitions in a Dictionary

To add dictionary entries:

1 Select the new dictionary in the *Available dictionaries* list and press *Modify*. The *Word Dictionary* dialog box is displayed. The dictionary name is displayed at the top of the dialog box followed by the list of words contained in the dictionary.

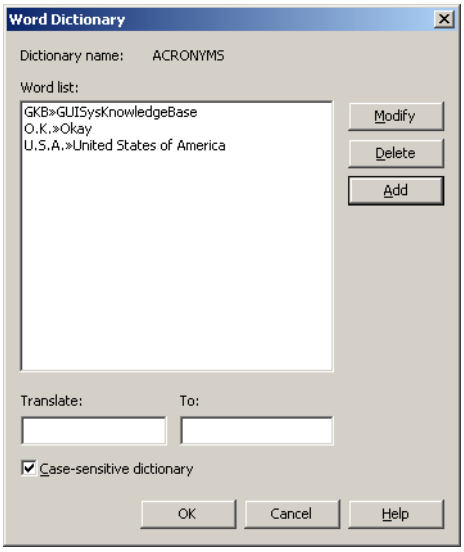


Figure 27. Word Dictionary dialog box

2 Add definitions to the dictionary by entering values in the *Translate* and the *To* fields. In the *Translate* field enter the word or phrase to be translated. In the *To* field, type how you would like these words represented. Click *Add* to add the new entry to the dictionary. When all entries are completed, click *OK*.

Note: The dictionary can be defined as case-sensitive or not case-sensitive by toggling the *Case-sensitive dictionary* check box. User dictionaries may also be accessed through the formats.ini file.

Part III. About Window Layouts

Window Layouts define how a host screen is converted to a GUI window. During the conversion process, ACE automatically suggests a GUI window for each Subapplication based on inherent Window Layouts. To achieve a particular effect or look, the developer may want to change the presentation suggested by the ACE converter.

A Subapplication's GUI window may be changed locally in Design View, but this does not effect the global look of the Application, thus making local changes necessary on all Subapplications. To avoid making local changes on all Subapplications, the developer can design new Window Layouts suited to the look of a particular Application.

New Window Layouts are stored in the KnowledgeBase. Window Layouts are applied when processing new Subapplications in the *New Subapplication* wizard.

Window Layouts consist of Selection Definitions paired with Function Definitions.

Chapter 5. Window Layouts

A Window Layout is a set of rules that determine the layout of controls and floating Representation Definitions on the GUI window.

An Application may make use of different Window Layouts to achieve different looks for different types of screens; one kind of Window Layout may be used for screens dealing with inventory, a different layout for order form screens, and yet another Window Layout for administrative task screens.

A Window Layout is defined by a series of control editing instructions that are performed in the order that they appear in the Window Layout definition.

A control editing instruction is a Selection Definition paired with a Function Definition. Selection Definitions select controls and Function Definitions perform editing functions such as positioning, alignment, spacing, and font control. When a Window Layout is applied to Subapplications, these instructions ensure that the GUI windows have a uniform look and feel.

This chapter describes:

- Accessing the Window Layout Definitions Manager
- Managing Window Layouts
- Creating New Window Layouts

Accessing the Window Layout Definitions Manager

Access the *Window Layout Definition* manager through the *Window Layouts* option in Design View or through the *Define* menu in the KnowledgeBase.

To open the *Window Layout Definitions* manager in Design View:

1 From the *Design* menu, select *Window Layouts*.The *Window Layouts Definitions* manager opens.

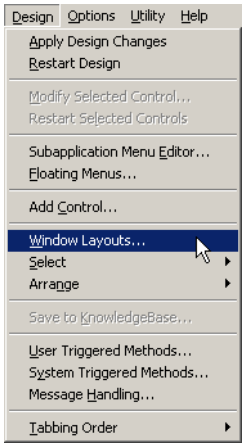


Figure 28. Window Layouts menu item in ACE

To open the *Window Layout Definitions* manager in the KnowledgeBase:

1. Open the *KnowledgeBase*.
2. From the *Define* menu, select *Window Layouts*. The *Window Layout Definitions* manager opens.

Managing Window Layouts

The *Window Layout Definitions* manager displays a list of Window Layout definitions that are stored in the KnowledgeBase. New Window Layout definitions written by the developer are added to this list.

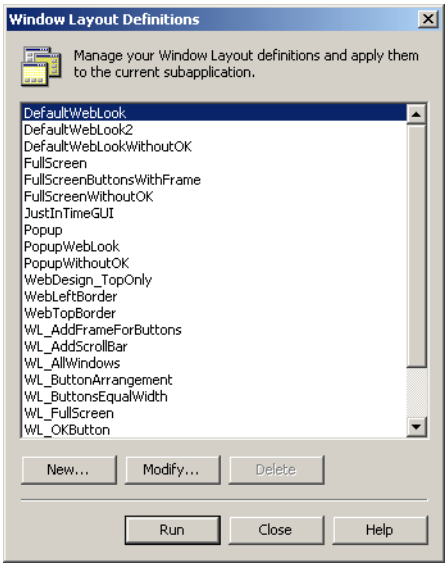


Figure 29. Window Layout Definitions manager

Click the option buttons in the Window Layout Definitions manager to create, edit, and run Window Layouts.

Table 9. Window Layout Definitions manager buttons

Option	Description
New	Opens the *Window Layout Definition* dialog box, where new Window Layouts are created.
Modify	Opens the *Window Layout Definition* dialog box to add, remove, and edit instructions from the *Instructions list* of existing Window Layouts.
Delete	Deletes the selected Window Layout. Deleting Window Layouts removes them from the KnowledgeBase. Only user-defined layouts can be deleted. Layouts provided by ACE are protected and cannot be deleted.
Run	Applies the selected Window Layout definition to the open Subapplication window.
Close	Accepts the current data and saves it to the KnowledgeBase.

Creating New Window Layouts

A series of instruction lines, organized in the *Instructions List* area of the *Window Layout Definition* dialog box, comprise a Window Layout. Each instruction line consists of a Selection Definition and a Function Definition. The instructions are applied to a Subapplication in the order that they appear in the list.

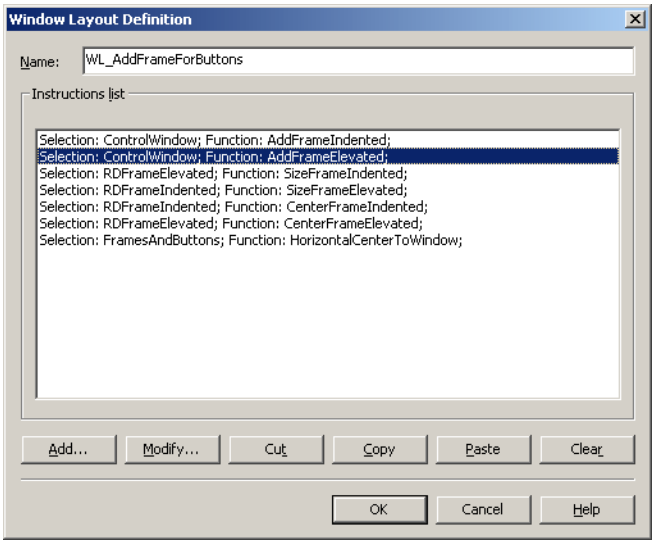


Figure 30. Instructions list area of the Window Layout Definition dialog box

The option buttons in the *Window Layout Definition* dialog box are used to create new Window Layouts, modify existing Window Layouts, and organize the instructions list.

Table 10. Window Layout Definition dialog box buttons (Sheet 1 of 2)

Option	Description
Add	Opens the *Control Editing Instruction* dialog box where you can select Selection and Function Definitions to add to the Window Layout instruction list.
Modify	Opens the *Control Editing Instruction* dialog box. Selection and Function Definitions can be modified, or new Selection and Function Definitions can be written, and added to the Window Layout instruction list.
Cut	Cuts the selected instruction from the list. After *Cut*, *Paste* is enabled.

Table 10. Window Layout Definition dialog box buttons (Sheet 2 of 2)

Option	Description
Copy	Creates a copy of the instruction line selection and places it on the internal clipboard. After using *Cut*, *Paste* is enabled.
Paste	Pastes an instruction from the internal clipboard into the instruction list. The instruction is pasted above the highlighted instruction. If no instruction is highlighted, it is placed at the end of the list.
Clear	Removes the highlighting from an instruction line.

To create a new Window Layout:

1. In the *Window Layout Definitions* manager, press *New.* The *Window Layout Definition* dialog box is displayed.
2. Type a name for the new Window Layout. The name must begin with a letter and can include letters, numbers, and underscores. Do not use blanks as separators. If the name is already in use, the original definition is overwritten when storing the new Window Layout.
3. Press *Add*. The *Control Editing Instruction* dialog box opens.
4. Choose a Selection Definition from the list in the left pane and a Function Definition from the list in the right pane by clicking on the definitions.
5. Click *OK*. The *Control Editing Instruction* dialog box closes. The selections are added to the instructions list.
6. Repeat steps 4 - 5, adding instruction lines until the Window Layout definition is complete.
7. Click *OK*. The new Window Layout is added to the KnowledgeBase and appears in all Window Layout lists. Pressing *Cancel* closes the *Window Layout Definition* dialog box without storing any information.

Control Editing Instruction Dialog Box

The *Control Editing Instruction* dialog box is used to add instructions to the Window Layout.

To create an instruction, a Selection Definition is chosen from the *Selection* list and paired with a Function Definition from the *Function* list.

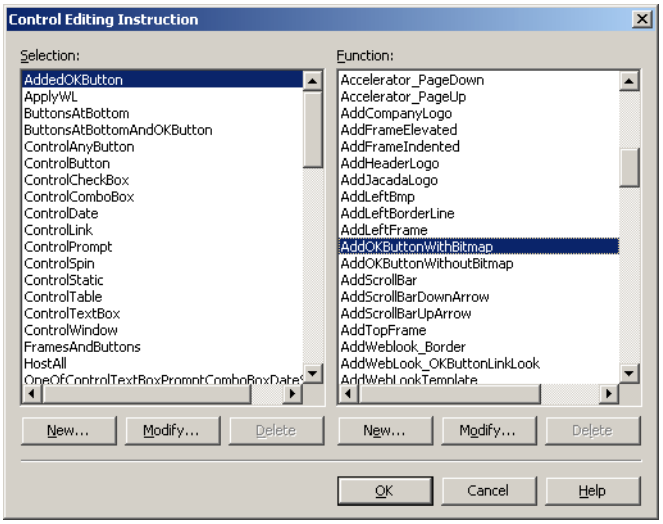


Figure 31. Control Editing Instruction dialog box

The *Selection Definition* dialog box and the *Function Definition* dialog box can be accessed from the *Control Editing Instruction* dialog box by selecting *Modify* or *New* or double-clicking on an item in the *Selection* or *Function* lists.

Table 11. Control Editing Instruction dialog box buttons (Sheet 1 of 2)

Option	Description
New	Opens the *Selection Definition* or *Function Definition* dialog boxes.
Modify	Opens the *Selection Definition* or *Function Definition* dialog boxes to the selected definition.
Delete	Deletes the selected definition.
OK	Places the instruction in the *Window Layout Definition* instruction list.

Table 11. Control Editing Instruction dialog box buttons (Sheet 2 of 2)

Option	Description
Cancel	Cancels the selections and closes the *Control Editing Instruction* dialog box.

Attaching Window Layouts to Subapplications

Window Layouts are attached to Subapplications using the *Layouts* dialog box which is accessed through the *Subapplication* menu. The *Subapplication* menu is available in all Views.

The *Subapplication* field in the *Layouts* dialog box displays the open Subapplication's name. To access a different Subapplication, open the *Subapplication* combo box. When a new Subapplication is selected, ACE saves all current information.

To attach a Window Layout:

1 From the *Subapplication* menu, choose *Layout*. The *Layout*s dialog box opens.

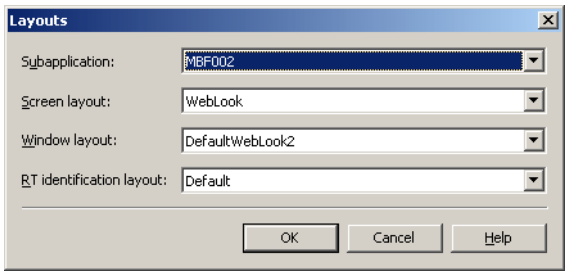


Figure 32. Layouts dialog box

2 Select a Subapplication from the *Subapplication* drop down combo box. Skip this step if working on the open Subapplication.

3 Select a layout from the *Window Layout* combo box.

4 Press *OK*. The *Layouts* dialog box closes.

5 Press the *Apply Changes* button in Design View to view the result of applying the selected Window Layout to the open Subapplication.

Window Layout Example

Create a Window Layout that adds a frame at the left side of the control window and places the company logo and FKey buttons in the frame.

The original Subapplication derived from the analyzed screen image:

Figure 33. A subapplication derived from an analyzed screen image

The new Window Layout definition:

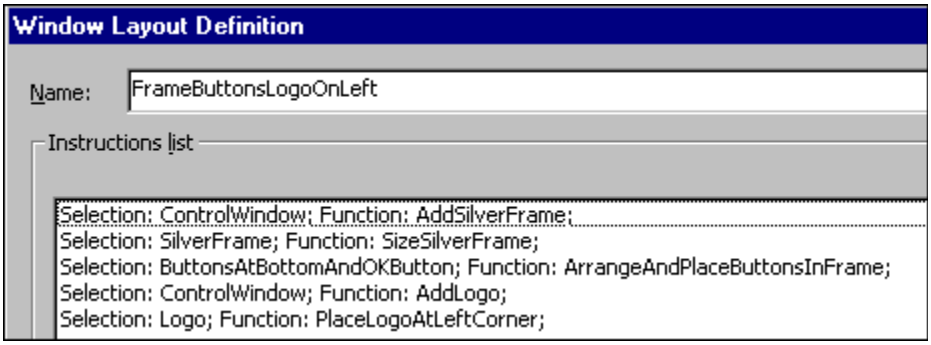


Figure 34. A new Window Layout definition

The Subapplication after applying the new Window Layout:

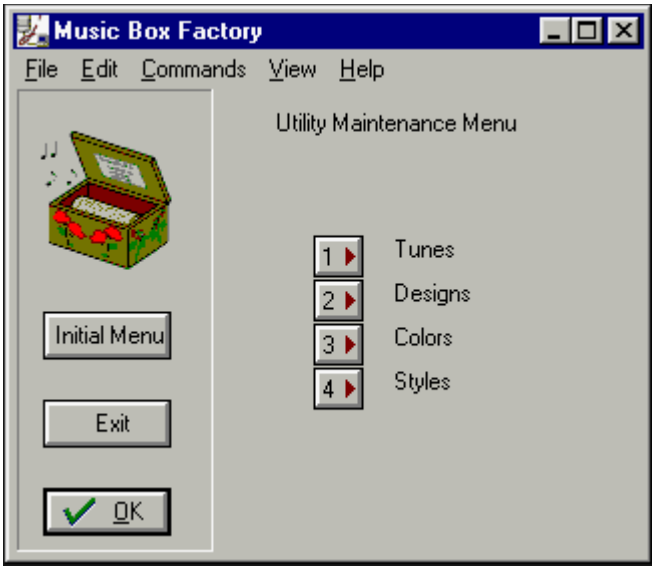


Figure 35. A new subapplication after the application of a new Window Layout

The Function Definitions that can be used to create this new Window Layout are not limited to the Function Definitions in this example. The versatility of Function Definitions provide the developer with many possibilities. Existing Window Layouts may be modified to conform to a design concept.

Writing Layouts to an *.ini File

Some existing applications have Window Layouts written to the specific.ini file.

To write the Subapplication layouts to the specific.ini file, add the following setting to the guisys.ini file:

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

The default value is 0 and means that the layouts will not be written to the *.ini file.

The layouts are written to the specific.ini file, but the setting is added to the guisys.ini file.

To read the layouts from the *.ini file, add the following setting to the library's specific.ini file:

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

The default value is 0 and means that the layouts will be read from the database.

Chapter 6. Selection Definitions

A Selection Definition delineates criteria used to select a control or a group of controls. Selection Definitions are defined in the *Selection Definition* dialog box and saved in the KnowledgeBase. Selection Definitions are used as stand alone when working on local modifications in Design View, and are paired with Function Definitions to create Window Layout instruction lines.

The dialog boxes used to create Selection Definitions are nested inside Design View.

Accessing the Selection Definitions Manager

Access the *Selection Definitions* manager through the *Design* menu in Design View or through the *Define* menu in the KnowledgeBase.

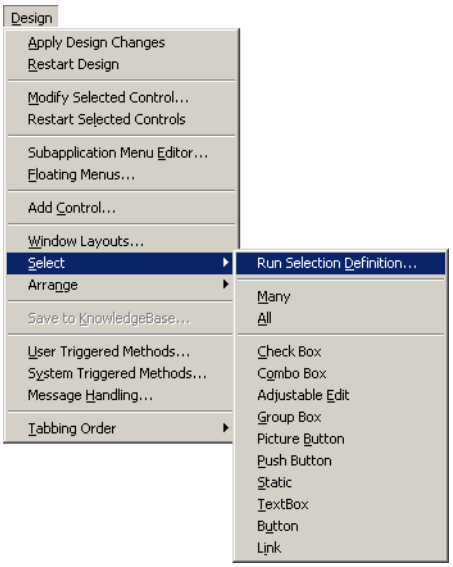


Figure 36. Accessing Selection Definitions through the Design menu in ACE

To open the *Selection Definitions* manager in Design View:

1 From the *Design* menu, choose *Select* > *Run Selection Definition*.

To open the *Selection Definitions* manager in the KnowledgeBase:

1 Open the KnowledgeBase.
2 From the *Define* menu choose *Selection Definitions*. The *Selection Definitions* manager opens.

Managing the Selection Definitions List

The *Selection Definitions* manager displays a list of the Selection Definitions that are stored in the KnowledgeBase. New definitions that are written by the developer are added to this list.

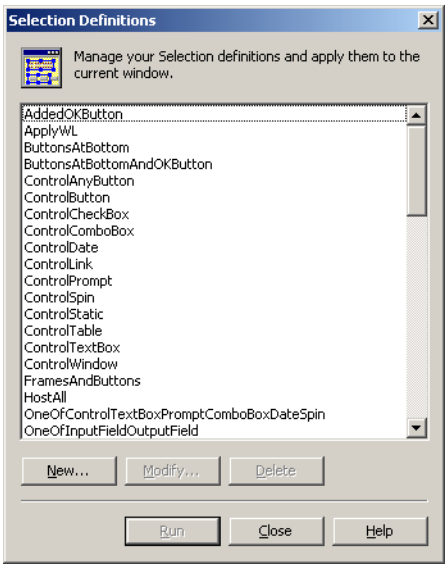


Figure 37. Selection Definitions manager

Table 12. Selection Definitions manager buttons

Option	Description
New	Opens the *Selection Definitions* dialog box, where new Selection Definitions are written.
Modify	Opens the *Selection Definitions* dialog box, displaying the Selection Definition highlighted in the manager. Modifications are saved in the KnowledgeBase.
Run	Applies the highlighted definition to the selected control(s)
Delete	Deletes the highlighted Selection Definition. Only developer-defined definitions or definitions that were modified can be deleted.
Close	Accepts the current data and saves it to the KnowledgeBase. The definition is not run.

Writing New Selection Definitions

The Selection Definition feature allows automatic selection of controls designated in the definition.

To write new Selection Definitions, in the *Selection Definitions* manager, press *New*. The *Selection Definition* dialog box is displayed.

1. Type a name for the Selection Definition. The name must begin with a letter and can include letters, numbers, and underscores. Do not use blanks as separators.
2. Choose a Selection Definition type from the drop down list.

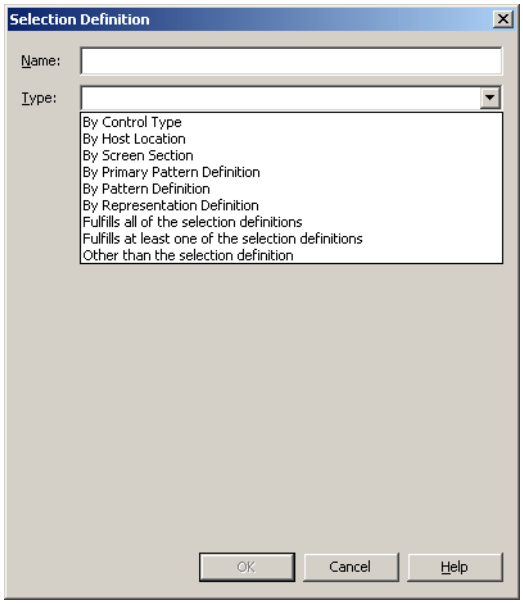


Figure 38. Choosing the Selection Definition type

3. Enter the information required by the *Type* selection.
4. Press *OK*. The new Selection Definition is stored in the KnowledgeBase and appears in the list displayed in the Selection Definitions manager.

Note: The *OK* button is not enabled until information is entered in both the **Name** and the *Type* fields.

Selection Definition Types

The parameters used to define a given set of controls are divided into categories. To view the categories available for the selection definition process, open the *Type* combo box. Each definition type prompts dedicated sections in the *Selection Definition* dialog box.

The table below provides short descriptions of each Selection Definition type.

Table 13. Selection Definition types (Sheet 1 of 2)

Type	Selection
By Control Type	Select controls by control type from the *Control Type* drop down list.
By Host Location	Select controls according to their location on the host screen.
By Screen Section	Select all the controls within a screen section, chosen from the Screen Section list.
By Primary Pattern Definition	Select controls according to the primary Pattern Definitions. Primary Pattern Definitions in a Subapplication can be seen in Layout View.
By Pattern Definition	Select controls according to the Pattern Definitions. Pattern Definitions in a Subapplication can be seen in Analysis View.
By Representation Definition	Select controls according to their Representation Definition. Representation Definitions in a Subapplication can be seen in Design View.
Fulfills all of the selection definitions	Use multi-parameter Selection Definitions to select controls. Only controls that satisfy all the parameters are selected.
Fulfills at least one of the selection definitions	Use multi-parameter Selection Definitions to select controls. Only controls that satisfy at least one parameter are selected.

Table 13. Selection Definition types (Sheet 2 of 2)

Type	Selection
Other than the selected definition	Use multi-parameter Selection Definitions to select controls. Selects all the controls on the window except for the ones generated from this Selection Definition. Describes which controls should not be selected.

Note: Definitions in the following examples have been created solely for the purpose of example.

By Control Type

Selecting *By Control Type* prompts a drop down list. Press the arrow key to view the control types.

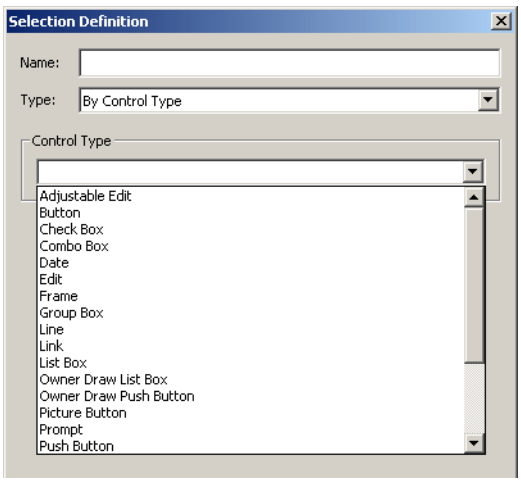


Figure 39. By Control Type dialog box

To write a new control type Selection Definition:

1. Go to the *Selection Definitions* manager.
2. Click *New*. The *Selection Definition* dialog box opens.
3. Click the arrow to display the *Type* list and select *By Control Type*.The selection appears in the *Type* field.
4. Click the arrow to display the *Control Type* list. Click on one of the control types. The list closes and the selection is highlighted in the *Control Type* box.

5 Assign a name to the new Selection Definition.
6 Click *OK*. The new Selection Definition is added to the Selection Definitions list.
7 To test the Selection Definition, go to the *Selection Definitions* manager and press the *Run* button. The selected controls are displayed with handles on the GUI window.

By Host Location

Controls can be selected according to their location on the host screen. To create a Selection Definition based on the host screen location, choose the *By Host Location* option in the *Selection Definition* dialog box.

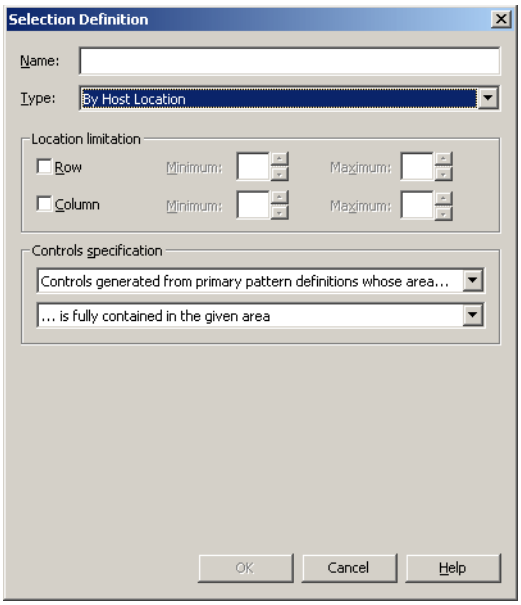


Figure 40. By Host Location

The *Location Limitation* section is used to specify the precise location of controls on the host screen. The *Colors Specification* section further refines the selection.

Location Limitation

The **By Host Location** selection type identifies controls which result from the analysis of a certain area on the host screen. To select controls by their location on the host, the relevant host screen area must be defined.

The host screen area is defined by the rows and columns and the area borders are limited by minimum and maximum settings.

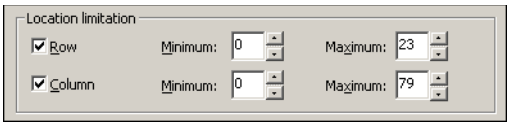


Figure 41. Location limitations of a host screen

Table 14. Location limitations of a host screen

Option	Description
Row	Define an area in terms of rows. The selected area begins with the row number indicated in the *Minimun* spinbox and ends with the row number indicated in the *Maximum* spinbox. If columns are not specified, the rows include the entire width of the screen.
Column	Define an area in terms of columns. The selected area begins with the column number indicated in the *Minimum* spinbox and ends with the column number indicated in the *Maximum* spinbox. If rows are not specified, the columns include the entire height of the screen.

Controls Specification

Two drop down lists contain clauses to further specify the control selection. The combination of the two clauses constitute the control specification.

The first drop down list pertains to controls and their origin on the host screen.

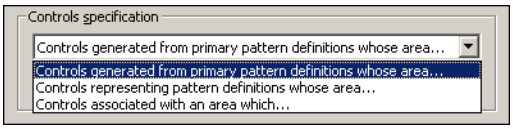


Figure 42. Control specification drop down list

Table 15. Control specification drop down list options

Option	Description
Controls generated from primary pattern definitions	This clause refers to the physical area on the host that the primary Pattern Definition, which yields the controls representing the primary Pattern Definition, occupies.
Controls representing pattern definitions	This clause refers to the physical area on the host that the Pattern Definition, which yields the control representing the Pattern Definition, occupies.
Associated with an area	Every control yielded by the analysis process is associated with some area on the host screen. This area is the pattern highlighted in red in the Pattern Definition Tree dialog box. This clause refers to the physical area on the host that is parallel to the red highlighted portion in the Pattern Definition tree.

The second dialog box is concerned with the relationship between the area that produces the control and the area delineated in the *Location limitation* section.

The area responsible for producing the control (control area) may interact with the given area as designated in the *Location limitation* section in the following ways:

- The control area may be fully contained in the given area.
- The control area may begin in the given area.
- The control area may intersect with the given area.
- The control area may end in the given area.

By Screen Section

Screen sections demark areas on the host screen where specific Pattern Definitions are applied during analysis.

Screen sections can be used as criteria to select controls. Choosing the *By Screen Section* option displays the *Section Definition* list box. The list box displays all the screen sections in the KnowledgeBase.

Controls generated in the area delimited by the section are selected according to the chosen screen definition.

Clicking *View,* opens the *Primary Section Definition* dialog box where Section Definitions can be modified.

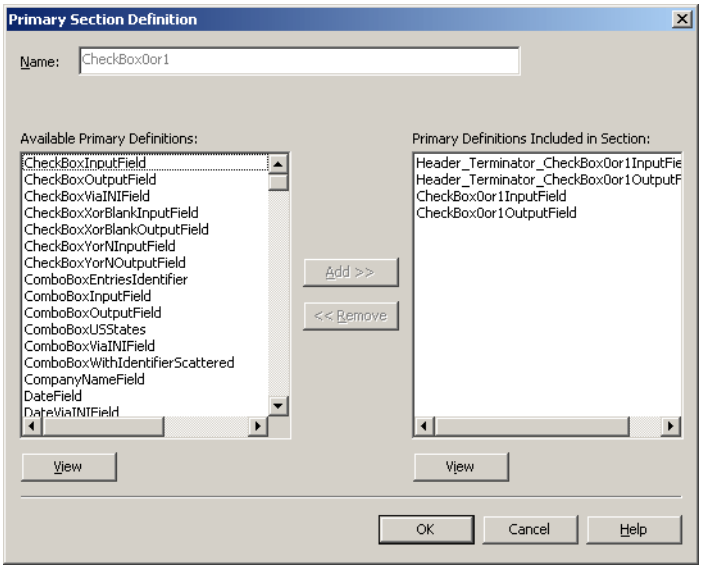


Figure 43. By Screen Section

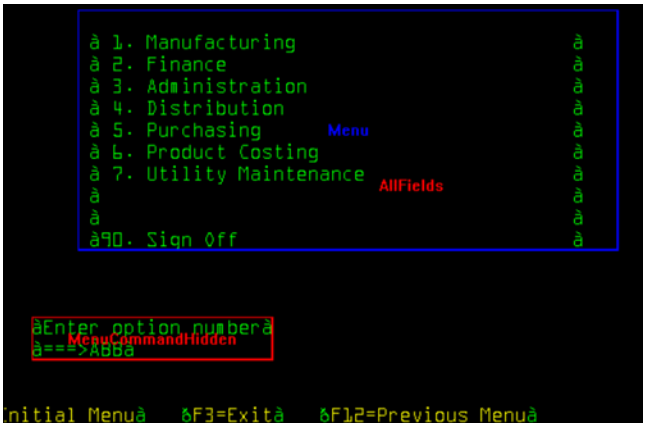
> **Note:** Changes made to a primary section are added to the KnowledgeBase and impact the analysis of all screens using a layout that includes this section.

To select controls that are located within a particular screen section:

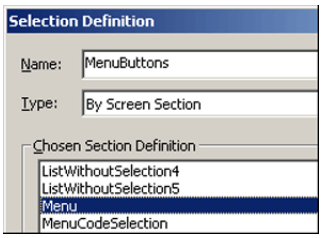
1. Chose *By Screen Section* the Selection Definition type.
2. Highlight the name of the screen section in the *Chosen Screen Section* list.
3. Enter a name for the new Selection Definition the *Name* edit box.
4. Press *OK*.

Example 7. By screen section

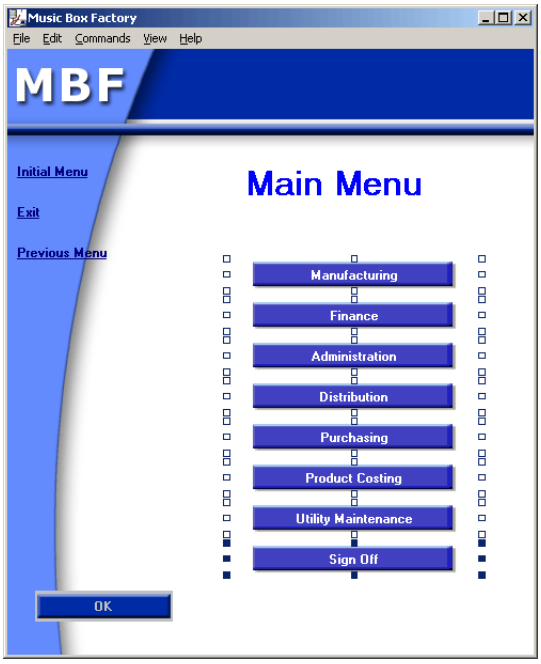
The buttons positioned in the middle of the window below were analyzed using the *Menu* section.



The following Selection Definition was written:



to select the buttons analyzed by *Menu*

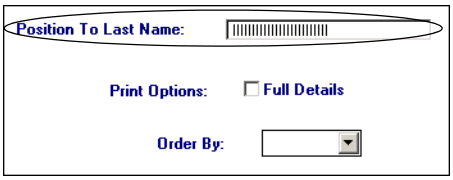


By Primary Pattern Definition

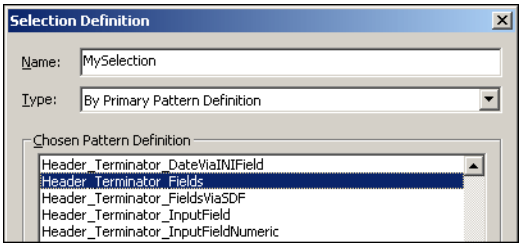
To write a Selection Definition according to a primary Pattern Definition:

1. Choose the *By Primary Pattern Definition* option from the *Type* list.
2. Select a primary Pattern Definition from the *Chosen Pattern Definition* list. Clicking *View* opens the KnowledgeBase to the selected primary Pattern Definition, which can then be viewed or modified.

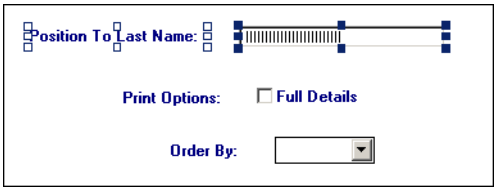
Example 8. By primary pattern definition



Select the Header and text box, which are recognized by the primary Pattern Definition named ***Header_Terminator_Fields***.



When this Selection Definition is run, the controls identified by ***Header_Terminator_Fields***. are selected.



By Pattern Definition

Pattern Definitions can be used to identify controls for selection. Controls are displayed on the GUI window through Representation Definitions, which represent Pattern Definitions.

To write a Selection Definition according to a Pattern Definition:

1. Choose the *By Pattern Definition* option from the *Type* list.
2. Select a Pattern Definition from the *Chosen Pattern Definition* list. Clicking *View* opens the KnowledgeBase to the selected Pattern Definition, which can then be viewed or modified.
3. Enter a name for the selection in the *Name* field.
4. Press *OK*.

By Representation Definition

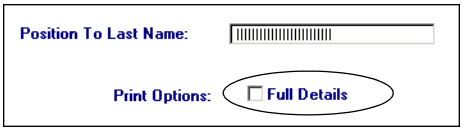
Representation Definitions are assigned to Pattern Definitions in the KnowledgeBase and represent the controls yielded by the analysis of the host screen. A Representation Definition name generally consists of a Pattern Definition followed by a phrase expressing the control that represents the Pattern Definition. The following is an example of a Representation Definition:

```
Header_Terminator_CheckBoxYorNIdentifier.CheckBox
```

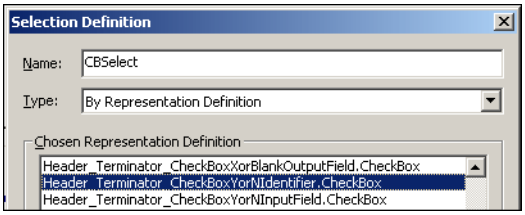
To write a Selection Definition according to a Representation Definition:

1 Choose the *By Representation Definition* option from the *Type* list.
2 Select the Representation Definition from the *Chosen Representation Definition* list.
3 Type a name in the *Name* field.
4 Press *OK*.

Example 9. By representation definition



Select the check box associated with the Representation Definition ***Header_Terminator_CheckBoxYorNIdentifier.CheckBox***.



Multi-Parameter Selection Definitions

The *Fulfills all of the selection definitions* and *Fulfills at least one of the selection definitions* options allow Selection Definitions that include more than one selection parameter. Multi-parameter Selection Definitions appear in the *Type* list in the Selection Definition dialog box.

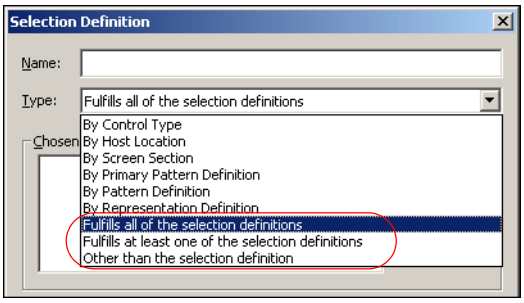


Figure 44. Multi-parameter selection definitions

There are two sections in the *Selection Definition* dialog box: *Chosen selection definitions* and *Selection definitions list*.

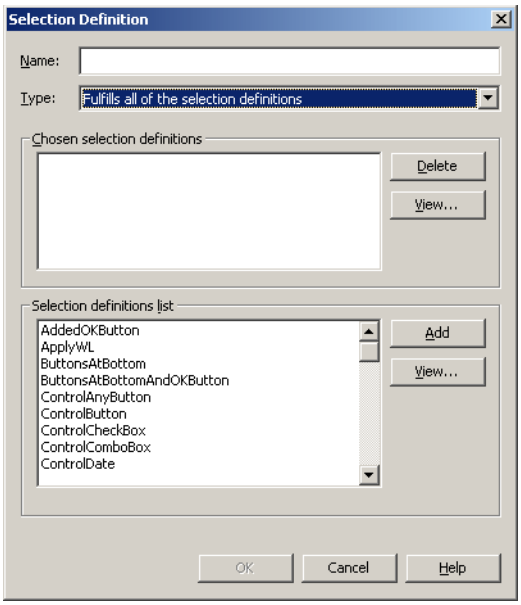


Figure 45. Selection Definition dialog box

The buttons are used to:

Table 16. Selection Definition dialog box options

Option	Description
Add	Add the Selection Definition highlighted in the *Selection definitions list* to the *Chosen selection definitions* list box.
Delete	Delete the Selection Definition highlighted in the *Chosen selection definitions* list box.
View	Displays the *Selection Definition* dialog box for the highlighted Selection Definition. The Selection Definition can be modified.

To create multi-parameter Selection Definitions:

1. Choose *Fulfills all of the selection definitions* or *Fulfills at least one of the selection definitions* from the *Type* list.
2. Choose a definition in the *Selection definitions list*.
3. Click *Add*. The definition is added to the *Chosen Selection Definitions*.
4. Repeat steps 3 and 4 until the Selection Definition is complete.

5 Enter a definition name in the *Name* field.

6 Click *OK*. When this Selection Definition is run, controls conforming to the definitions appearing in the *Chosen Selection Definitions* list are selected on the GUI window.

Other Than the Selection Definition

The *Other than the selection definition* option selects all controls on the window except for the chosen Selection Definition. This Selection Definition describes which control should *not* be included in the selection.

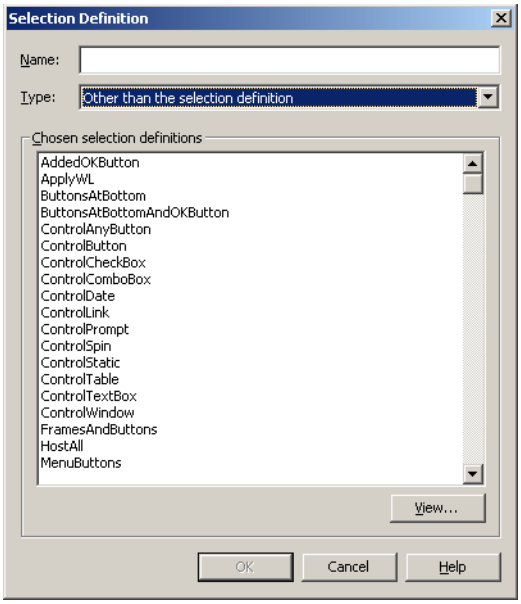


Figure 46. Other than the selection definition option

Example 10. Other than the selection definition

Select all controls which are not buttons:

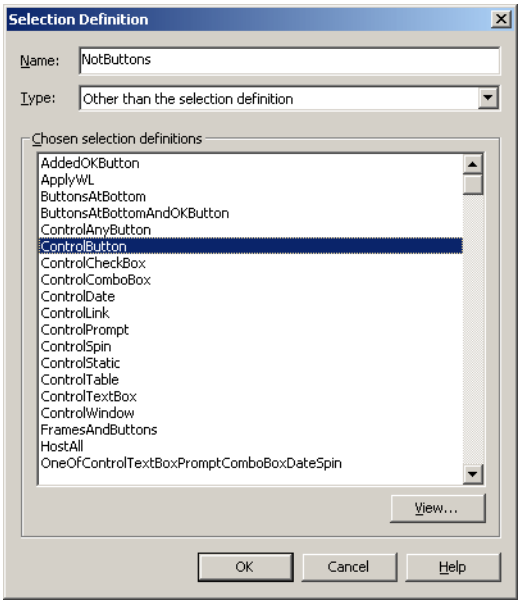
1 Select *Other than the selection definition* from the *Type* list.

2 Choose the Selection Definition that selects all the Buttons.

3 Enter a definition name in the *Name* field.

4 Press *OK*.

The Selection Definition looks like this:



Chapter 7. Function Definitions

A Function Definition edits a single control or a group of controls for placement, size, and spacing. Function Definitions are modified in the Function Definition dialog box and saved in the KnowledgeBase. Function Definitions are used as stand alone when working on local modifications in Design view, and are paired with Selection Definitions to create Window Layout instruction lines.

The *Function Definitions* manager displays a list of all default and developer written Function Definitions.

This section includes:

- Accessing the Function Definitions Manager
- Managing Function Definitionss
- Writing New Function Definitions

Note: In the following sections, windows were created solely for examples and do not reflect ACE presentations.

Accessing the Function Definitions Manager

Access the *Function Definitions* manager through the *Arrange* menu option in Design View or through the *Define* menu in the KnowledgeBase.

To open the Functions Definitions manager in Design View:

1. Select a control.
2. From the *Design* menu, select *Arrange* > *Run Function Definitions*;

   OR

   Right click on the selected control and choose *Run Function Definitions* from the shortcut menu.

   The *Function Definitions* manager opens.

To open the Functions Definitions Manager in the KnowledgeBase:

1. Open the KnowledgeBase.
2. From the *Define* menu select *Function Definitions*. The *Function Definitions* manager opens.

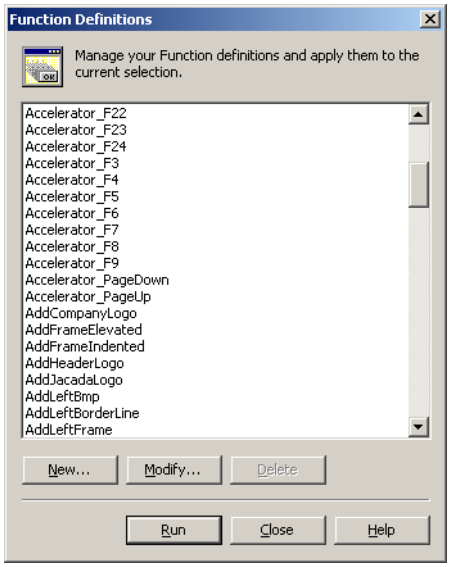


Figure 47. Function Definitions manager

Managing Function Definitions

The *Function Definitions* manager displays a list of the Function Definitions that are stored in the KnowledgeBase. New definitions that are written by the developer are added to this list.

Click the option buttons in the *Function Definitions* manager to write, modify, and run Function Definitions

Table 17. Function Definitions manager options (Sheet 1 of 2)

Option	Description
New	Opens the *Function Definition* dialog box, where new Function Definitions may be written.
Modify	Opens the *Function Definition* dialog box, where Function Definitions may be modified.
Run	Applies the highlighted definition to the selected control(s).
Delete	Deletes the selected Function Definition. **Note:** Only developer-defined definitions or definitions that were modified can be deleted.

Table 17. Function Definitions manager options (Sheet 2 of 2)

Option	Description
Close	Accepts the current data and saves it to the KnowledgeBase. No definitions are applied to the selected controls.

Writing New Function Definitions

The Function Definition feature determines how controls are presented, arranged, and configured on the GUI window.

To write new Function Definitions:

1. Press *New* in the *Function Definitions* manager. The *Function Definition* dialog box is displayed.
2. Type a name for the *Function Definition*.The name must begin with a letter and can include letters, numbers, and underscores. Do not use blanks as separators.
3. Choose a Function Definition type from the drop down list.

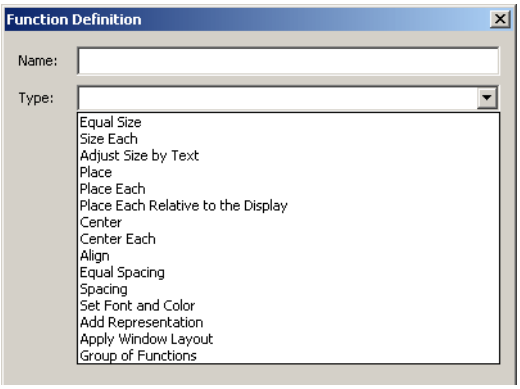


Figure 48. Choose a function definition type

4. Enter the information required by the *Type* selection.
5. Press **OK**. The new Function Definition is stored in the KnowledgeBase and appears in the list displayed in the *Function Definitions* manager.

Note: The *OK* button is not enabled until information is entered in both the *Name* and *Type* fields.

Function Definition Types

The criteria that defines how a control appears on the GUI window are:

- Size
- Placement
- Physical presentation - height, width, font, and color
- Spacing relative to the window and other controls

The criteria for formatting controls via Function Definitions are displayed in the *Type* drop down box in the *Function Definition* dialog box.

The table below provides short descriptions of each Function Definition type.

Table 18. Function Definitions manager options (Sheet 1 of 2)

Type	Function
Equal Size	Define the size of controls as defined by the size of a designated control within the selected group.
Size Each	Define the size of controls, within the selected group, using the pixel (display) unit as the standard of measurement.
Adjust Size By Text	Resize a control according to the text it contains.
Place	Move the selected controls to a specified location without changing their relative positions.
Place Each	Move all selected controls individually relative to a set of anchor controls that do not move.
Place Each Relative to the Display	Move all selected controls individually relative to the Subapplication window.
Center	Position a control or group of controls relative to the center of a second control or group of controls that were defined as the anchor controls. The repositioned controls retain their relative positions.

Table 18. Function Definitions manager options (Sheet 2 of 2)

Type	Function
Center Each	Move all selected controls individually relative to the center of a set of anchor controls.
Align	Align a group of controls either horizontally or vertically in relation to a designated anchor control.
Equal Spacing	Position the selected controls at equal intervals within a given area defined by the anchor controls.
Spacing	Space controls at predetermined distances from one another.
Set Font and Color	Assign font, text, and background color to a control.
Add Representation	Add a Floating Representation to the GUI window.
Apply Window Layout	Apply a Window Layout to selected controls.
Group of Functions	Combine multiple Function Definitions into a single definition. This definition, with its various commands, can then be applied to a selected control or group of controls.

Anchor controls

Function Definitions that position and space controls make use of anchor controls. Anchor controls are a single control or a group of controls that are designated as such when writing the Function Definition. The corners and center of an imaginary rectangle that encompasses the anchor controls are the points referenced by Function Definitions.

The Function Definitions that use anchor controls are:

- Place
- Place Each

- Center
- Center Each
- Align
- Equal Spacing
- Spacing

The Distance Section

The *Distance* section is common to the following types:

- Place
- Place Each
- Place Each Relative to Display
- Spacing

Distance is defined in display units or a percentage of a control that has been previously defined as a Selection Definition.

For placement type Function Definitions, use the *Horizontal* and *Vertical offset* spinboxes to define the offset in display units. Negative values may be assigned. Negative numbers indicate which direction controls are being moved.

Table 19. Distance section options

Option	Description
Horizontal Offset (positive numerals)	Positions the mobile control from left to right relative to the stationary control.
Horizontal Offset (negative numerals)	Positions the mobile control from right to left relative to the stationary control.
Vertical Offset (positive numerals)	Positions the mobile control down, relative to the stationary control
Vertical Offset (negative numerals)	Positions the mobile control up, relative to the stationary control

When writing a Spacing type Function Definition, the *Distance* section is used to space controls vertically, horizontally, or diagonally. To space controls vertically or horizontally, set the appropriate check box and use the spinbox arrows to set the number of display units.

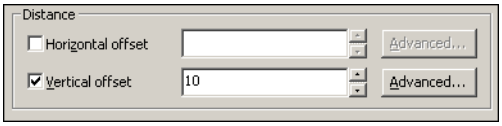


Figure 49. Distance section

To create diagonal spacing, set both check boxes and set the spinboxes to the desired distances.

The *Distance* section also provides access to the *Advanced Measurement* dialog box. The *Advanced Measurement* dialog box options are used to define distance using the dimensions of Selection Definitions.

The Advanced Measurement Dialog Box

The *Advanced Measurement* dialog box is common to the Function Definitions:

- Size Each
- Place
- Place Each
- Equal Spacing
- Spacing

The *Advanced Measurement* dialog box is used to determine the size or position offset of controls by setting absolute display units, a percentage of the height or width of controls selected by Selection Definitions, or by both percentage and display units.

Access the *Advanced Measurement* dialog box by clicking on *Advanced* in the *Function Definition* dialog box.

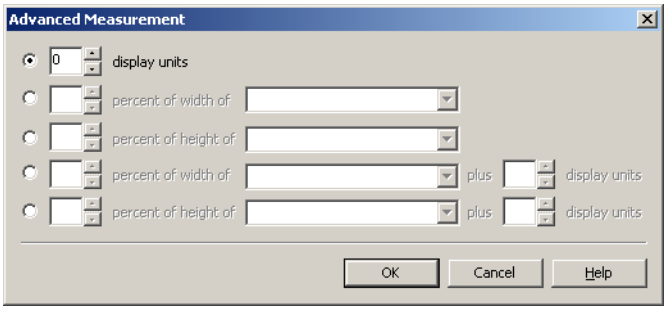


Figure 50. Advanced Measurement dialog box

To size or offset controls by display units (pixels):

1. Click the *display units* radio button.
2. Use the spinbox arrows to set the number of display units. Only positive numbers are allowed.
3. Click *OK*.

To size or offset controls by a percentage of a Selection Definition:

1 Click on the radio button beside the appropriate line. *Percent of width of* or *Percent of height of* is enabled. `100`, the default percentage, appears in the spinbox.

2 Open the combo box to view a list of Selection Definitions.

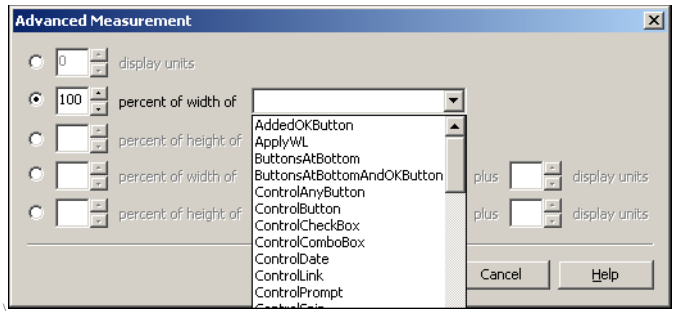


Figure 51. Advanced measurement options

3 Select the appropriate list entry. Use the spinbox to adjust the percentage of the control's height or width.

The last two lines in the dialog box use a combination of control percentages and display units. To format a control using this option, select a control and its percentage. Then use the additional spinbox to add display units to the dimensions.

Example 11. Advanced measurement

Create spacing with the width of an *Exit* button:

1 From the combo box, highlight the Selection Definition that selects the *Exit* button.

Figure 52. Advanced measurement - percent of width of

2 Optional: Use the spinbox arrows to change the percentage.

3 Press *OK*.

4 Assign the new Function Definition a name.

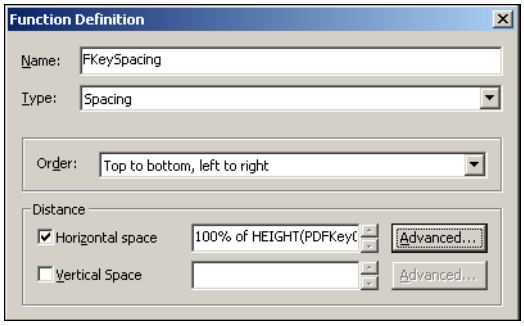


Figure 53. Naming the function definition

5 Press **OK**. The new Function Definition is stored in the KnowledgeBase and added to the Function Definitions list.

Equal Size

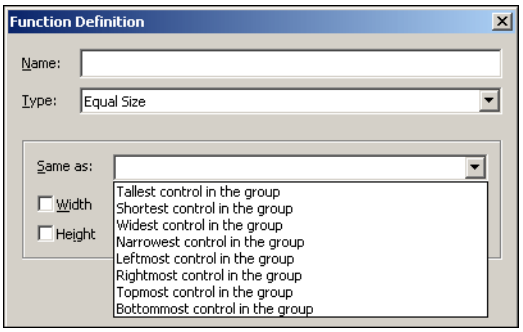


Figure 54. Function definition - Equal Size

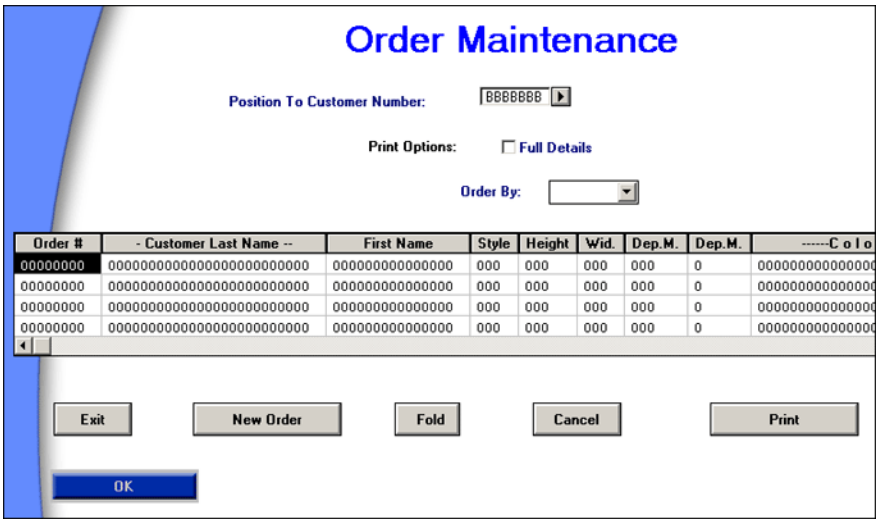
To set a uniform size for controls:

1 Select controls by dragging a rectangle or using a Selection Definition.
2 Open the *Function Definitions* manager.
3 Click *New*. The *Function Definition* dialog box opens.
4 Select *Equal Size* from the *Type* combo box.
5 Select a reference control from the *Same as* combo box.
6 Check *Width* and/or *Height* as appropriate.
7 Enter a Function Definition name in the *Name* field.
8 Click *OK*. The *Function Definition* dialog box closes. The new Function Definition is added to the list in the *Function Definitions* manager. When run, all selected controls are uniformly sized according the parameters set for this Function Definition.

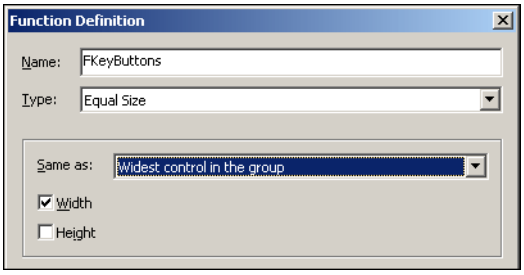
> **Note:** Width and height refer to the dimensions of the operator in the *Same as* field.

Example 12. Equal Size

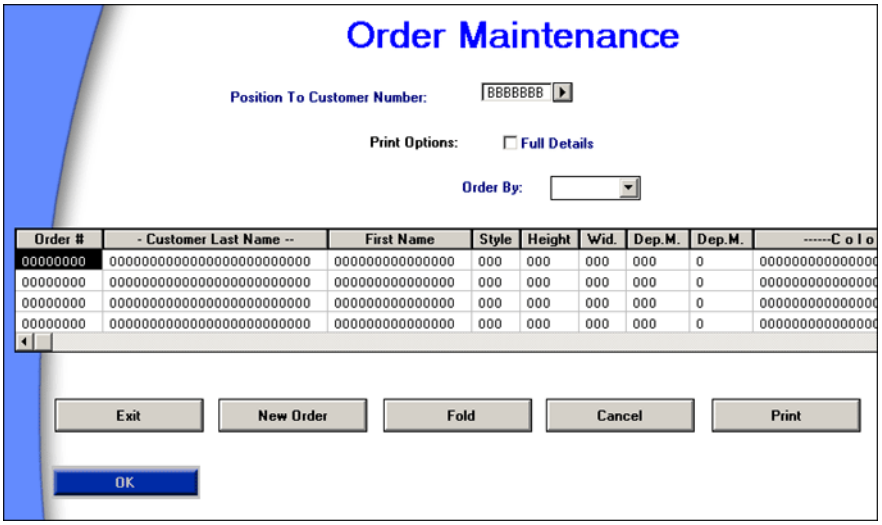
The GUI window prior to applying the Equal Sizing option:



The Function Definition:



The result of applying the Function Definition is:



Size Each

The Size Each type defines the size of controls using display units, the standard unit of measurement on a computer screen. The *Advanced Measurement* dialog box allows the developer to utilize a specified control to determine the size of the selected controls.

To create a Size Each Function Definition:

1. Go to the *Function Definition* dialog box and select the *Size Each* option.
2. Check the *Width* and/or *Height* check boxes. When the *Width* or *Height* check boxes are set, the spinboxes and the *Advanced* buttons are enabled. To determine the height and width of the Selection Definitions in display units, use the spinboxes until the desired height and width are reached. To use the dimensions of Selection Definitions to assign height and width, select the **Advanced** button and configure the appropriate settings.
3. Press *OK*. The *Function Definition* dialog box closes. The new Function Definition is added to the list in the *Function Definitions* manager.

Adjust Size By Text

The Adjust Size by Text type resizes controls to accommodate the text. This is useful when the control size is no longer appropriate due to a change in the font size.

Set the *Horizontal adjustment* and *Vertical adjustment* check boxes to enable their formatting options.

Table 20. Function definition - Adjust size by text options

Option	Description
No addition to width	The width ends immediately after the text, unless other parameters have been assigned.
Add display units to width	Add display units to the width using the spinbox.
Add percent of width to width	Increase the width by a percentage of its current width.
Add percentage of average character's width to width	Every hundred percent registered in the spinbox extends the width of the control a single character's width.

Note: The Adjust Size By Text option does not work on Lists.

Place

The Place type Function Definition allows moving the selected controls as a group, without altering their relative positions within the group. The group moves relative to anchor controls that remain stationary.

The *Relative place* section configures the coordinates of one control according to the coordinates of another control.

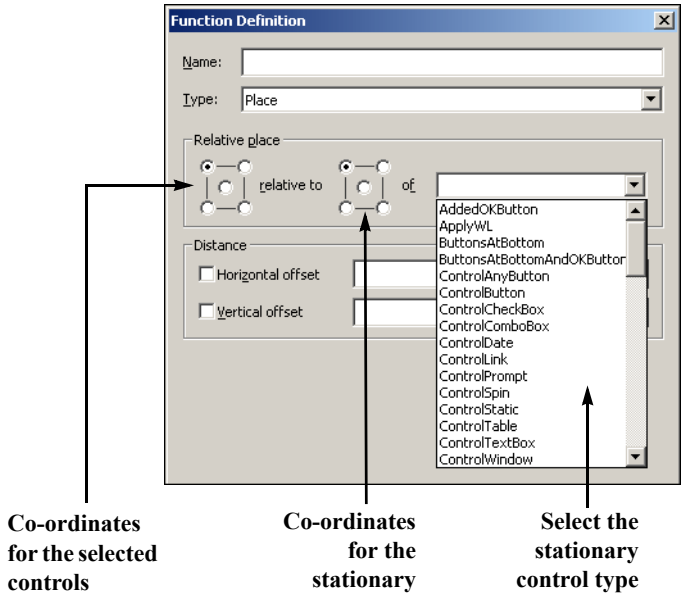


Figure 55. Function Definition - Place options

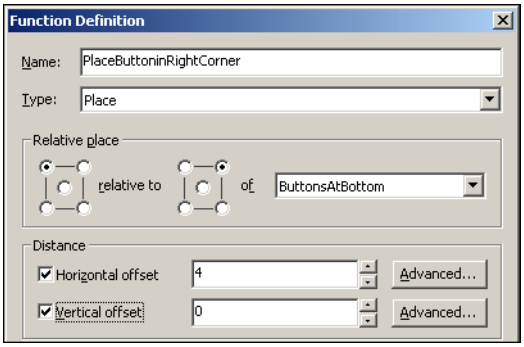
The squares in the *Relative place* section represent a control or a group of controls. The points on each corner and in the middle are coordinates of the control. Use these coordinates to define how controls are positioned in relation to one another. The upper left point represents the upper left corner of the control, the center point represents the center of the control, etc. When using groups of controls, the corner points and center point represent co-ordinates of the unseen rectangle that encompasses the group selection.

- The first square, in the *Relative place* section represents the selected control, which is moved and placed relative to the stationary control. Controls are selected manually in Design View or by running a Selection Definition.
- The second square represents the stationary control. The stationary control is selected from the list in the combo box.
- Use the *Distance* section to set the placement offsets for the selected controls.

Example 13. Function definition - Place option

Place the selected control, the *Next Menu* button, next to the stationary Previous *Menu* button. *Reminder:* The selected control is the control that moves.

The Function Definition looks like this:



The GUI presentation after applying the Place type Function Definition:

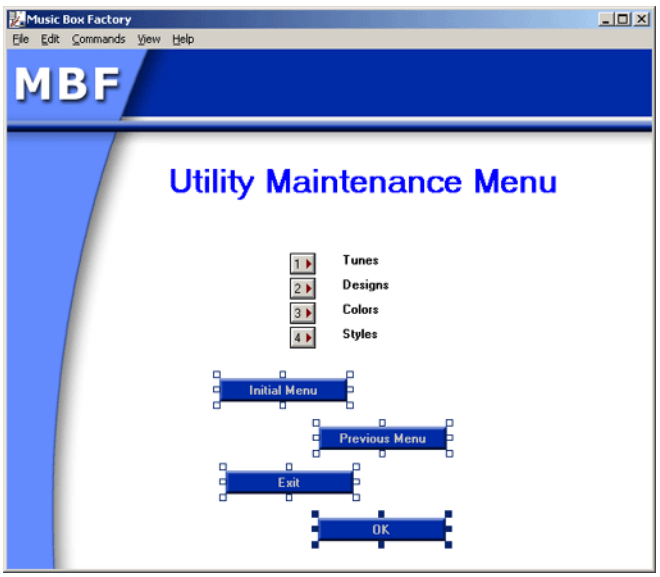
Note: The the entire GUI window can be used as a stationary control allowing controls to be placed in any of the corners or in the middle of the window. To place the selected control off-center or slightly away from a corner use the *Distance* section to define the offset.

Place Each

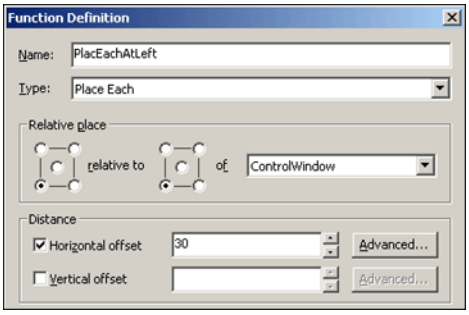
The Place Each type moves controls within the selected group individually. Controls move relative to anchor controls and are individually placed according to the coordinates defined in the *Relative place* section.

Example 14. Function definition - Place Each option

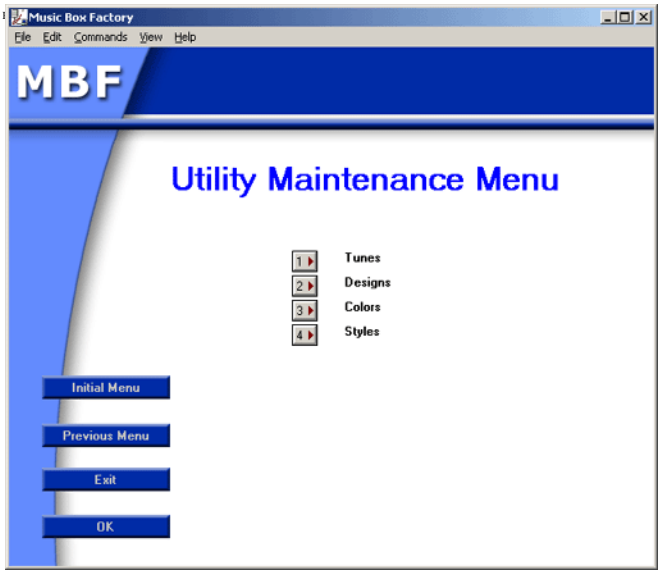
Use the Place Each option to place each menu button at a specified location.



Place Each treats each button as an individual control. When the following Function Definition is applied, each button is positioned along a defined parallel instead of staggered.



This Function Definition says: Take each of the selected controls and place them at the left, offset 30 units from the left edge of the GUI window.



Place Each Relative To the Display

This Function Definition type is used to place a control relative to the monitor screen. Generally, this option is applied solely to the control window.

In the *Relative place* section, indicate the coordinates of the window relative to the display. In the *Distance* section, adjust the *Horizontal* or *Vertical offset* to position the window in the display.

Note: In the *Relative place* section, the combo box is disabled and the option, *the display* is automatically assigned. Since the *Horizontal* and *Vertical offset* may only be set in terms of standard units, there is no *Advanced* button in this option.

Table 21. Function definition - Place Each Relative To Display options (Sheet 1 of 2)

Option	Description
Horizontal offset (positive numerals)	Moves the window from left to right on the monitor.

Table 21. Function definition - Place Each Relative To Display options (Sheet 2 of 2)

Option	Description
Horizontal offset (negative numerals)	Moves the window from right to left on the monitor.
Vertical offset (positive numerals)	Moves the window further down on the display.
Vertical offset (negative numerals)	Moves the window further up on the display.

Center

The Center type moves a control, or group of controls, and coordinates its placement to correspond to the horizontal or vertical center of the anchor controls.

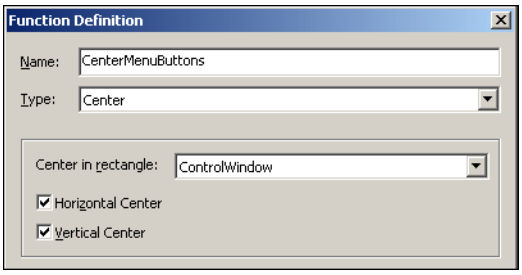
This option is useful for placing a group of controls in the center of the window. Similarly, this option can be applied to place a group of controls in the middle of a group box.

Note: The *Center* option is equivalent to using the *Place* option, selecting the center in both boxes, and setting the vertical and horizontal offset to zero.

Center in rectangle is used to designate the anchor controls and contains a list of Selection Definitions. Controls are centered according to the *Horizontal Center* and *Vertical Center* check boxes. If *Horizontal Center* is set, the group of controls is placed along the horizontal center of the group. If *Vertical Center* is set, the group of controls is placed along the vertical center. If both are set, the group of controls is placed directly in the center of the rectangle encompassing the group of anchor controls as defined the *Center in rectangle* field.

Example 15. Function definition - Center option

Place menu buttons in the center of the GUI window.



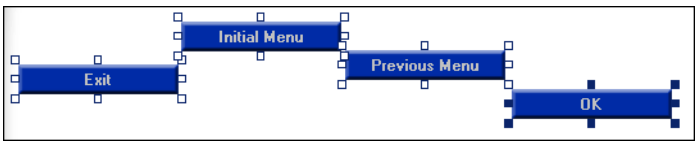
When this Function Definition is applied to the buttons yielded by the Menu screen section, they are automatically positioned in the window's center.

Center Each

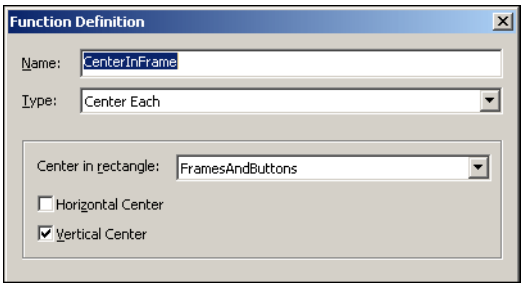
The Center Each type moves controls within the selected group individually. Controls move relative to anchor controls and are individually placed according to the defined coordinates.

Example 16. Function definition - Center Each option

The Center Each option places each button at a specified location.



Center Each treats each button as an individual control. When the following Function Definition is applied, the buttons are positioned along the defined parallel within the group box instead of staggered.



Applying the Function Definition results in the following GUI:

Align

The Align type Function Definition aligns a group of controls in relation to an anchor control. Controls can be aligned vertically or horizontally and according to the top, bottom, left, or right of the leader control.

Horizontal alignment moves the controls horizontally creating a column. Vertical alignment moves the controls vertically creating a row. The centers of the controls may be positioned along the vertical or horizontal center of the anchor control.

Alignment Type Section

Table 22 shows the options in the Align Type section.

Table 22. Function definition - Align Type option (Sheet 1 of 2)

Option	Description
Horizontal Alignment	
Left Sides	Controls are arranged in a column with their left borders aligned with the left anchor control border.
Horizontal Centers	Controls are positioned in a column with their centers aligned with the horizontal center of the anchor control.
Right Sides	Controls are positioned in a column, with their right borders aligned with the right anchor control border.
Vertical Alignment	
Tops	Controls are arranged in a row. The top edge of the controls are aligned with the top edge of the anchor control.

Table 22. Function definition - Align Type option (Sheet 2 of 2)

Option	Description
Vertical Centers	Controls are positioned in a row. Their centers are aligned with the vertical center of the anchor control.
Bottoms	Controls are arranged in a row. The bottom edge of the controls are aligned with the bottom edge of the anchor control.

Align To

The anchor control is chosen in the *Align to* combo box. Anchor criteria is defined by dimension or location.

- Dimensions: *Tallest*, *Shortest*, *Widest*, and *Narrowest*
- Anchor control: *Topmost*, *Bottommost*, *Leftmost*, and *Rightmost*

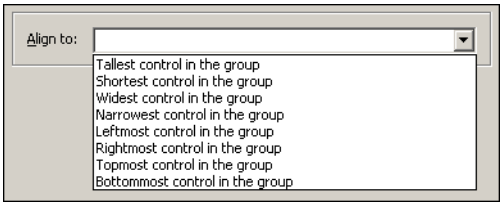
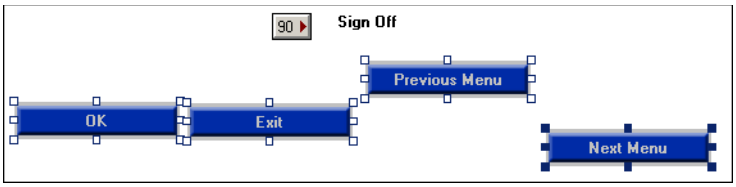


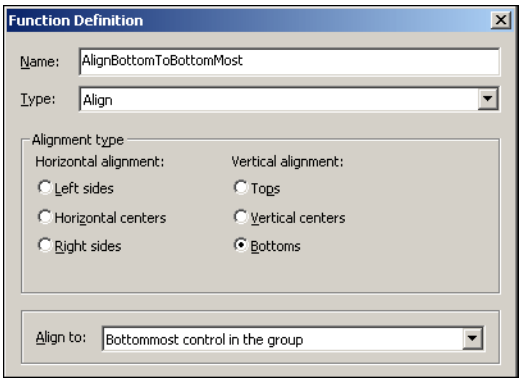
Figure 56. Align-to section

Example 17. Function definition - Align-to option

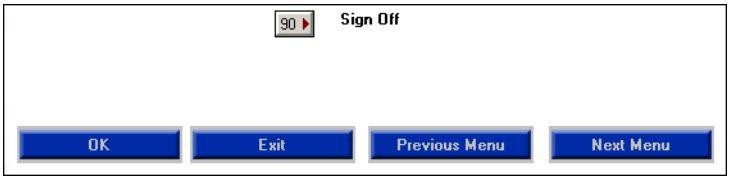
Align the staggered buttons.



This Function Definition aligns the bottom border of the selected buttons with the bottom border of the anchor control.



After applying this Function Definition the buttons are aligned according to the *Next Menu* button, which was the bottommost control:



Note: The Align option does not space the controls. To space the controls after alignment use the Spacing or Equal Spacing Function Definitions.

Equal Spacing

The Equal Spacing type positions controls at equal horizontal and/or vertical intervals within a specific area. The distance between the margins of the area and the first and last control in the group is determined by the *Horizontal* and *Vertical spacing* settings.

Spacing Type

In the *Spacing Type* section, which is common to both the Equal Spacing and Spacing Function Definitions, there are two combo boxes: *Order*, and *In rectangle*.

Order

The selection in the *Order* combo box determines the lead or priority control when applying a Spacing type Function Definition.

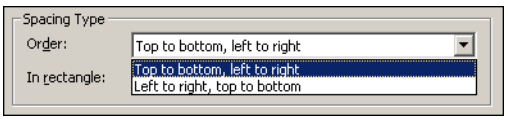


Figure 57. Order option

In Rectangle

The *In rectangle* option describes the area where the controls will be arranged. Controls may be spaced in relation to the entire GUI window or in a separate area, such as a group box.

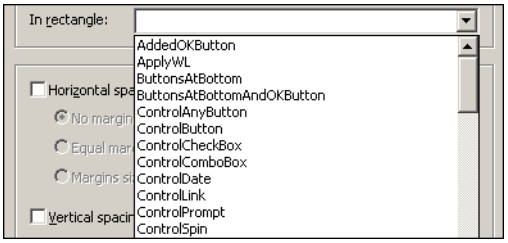


Figure 58. In Rectangle option

Horizontal and Vertical Spacing

Once the placement area is selected, the user may determine how controls should be spaced within that area.

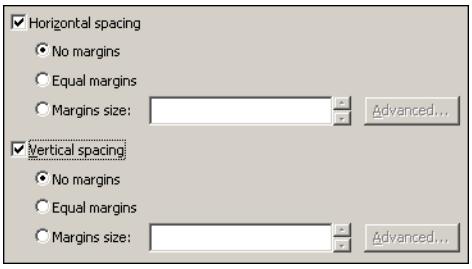


Figure 59. Horizontal and vertical spacing options

Table 23. Horizontal and vertical spacing options

Option	Description
Horizontal spacing	Space controls horizontally in a row.
Vertical spacing	Space controls vertically, one atop the other.

Note: If both the horizontal and vertical check boxes are set, the controls are aligned diagonally.

Table 24. Margin options

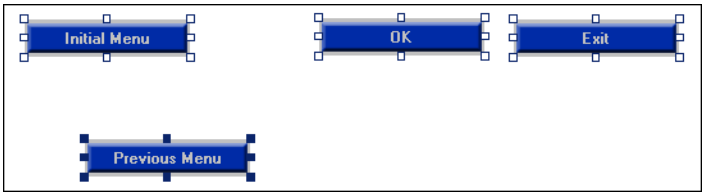
Option	Description
No Margins	The first control is situated immediately against the selected area's left border and the last control is positioned against the selected area's right border.
Equal Margins	The distance between controls is the same as between the first and last control and the area borders.
Margins Size	Margin size may be determined by display units, percentages of previously defined controls, or a combination of both. These options are available when clicking *Advanced*.

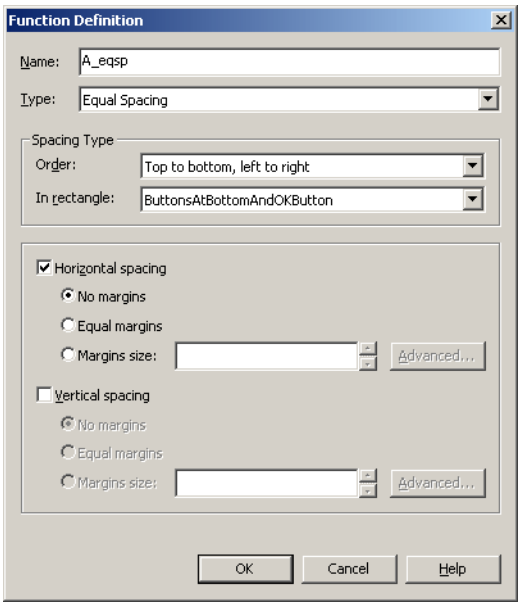
Note: There are no spinboxes next to the first two selections because the spacing is dictated automatically according to the controls and the allotted space.

Example 18. Horizontal and vertical spacing

Space the keys in the Command Keys section equidistant from one another.

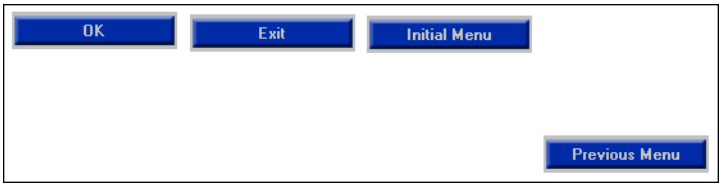
This is the original arrangement of the keys:





Using this Function Definition, the controls will be evenly spaced in the rectangular area. Since only the *Horizontal Spacing* check box is set, there is no vertical movement of the controls.

Applying the Function Definition results in the following GUI:



The last control is not aligned with the other buttons because the *Equal Spacing* option is not responsible for the alignment of controls. To align the last control with the other buttons, apply the *Align* definition.

Spacing

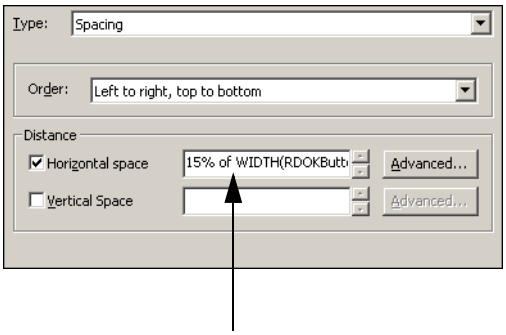
The Spacing type Function Definition determines the amount of space between individual controls. Spacing does not align controls; it places controls at a predetermined distance from one another.

Example 19. Spacing

Space the buttons evenly with the distance between the buttons set at 15 percent of the *OK* button.

Before:

Function Definition:



The value and the reference control were set in the *Advanced* dialog box

After:

Set Font and Color

The Set Font and Color type Function Definition is used to change the font, font color and/or background color.

Pressing the *Font and Color* button opens the *Font and Color* dialog box. The selections made in the dialog box are registered in the *Chosen font and color* edit box in the *Function Definition* dialog box.

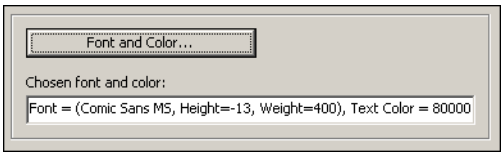


Figure 60. Set font and color

Add Representation

The Add Representation type creates a control representation on the window that does not result from a Pattern Definition and does not appear on the host.

Some uses of the Add Representation option:

- Create a button that immediately advances to another program
- Create a field that displays a company logo

Like Window Layout options, the presence and functionality of these representations may be applied throughout an entire Application.

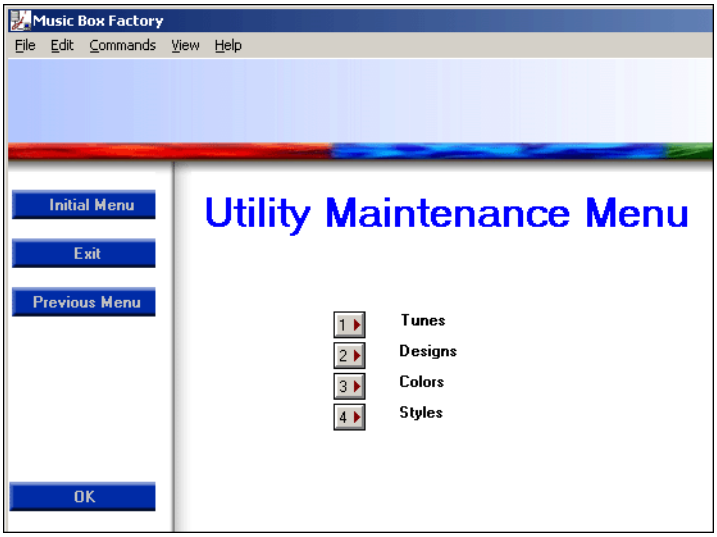
The *Representation Definition* button opens the *Representation Definitions List* dialog box. Choose a Representation Definition and click *OK*. The *Representation Definition List* dialog box closes and the *Chosen Representation Definition* field displays the Representation Definition chosen from the *Representation Definitions List*.

Example 20. Add representation

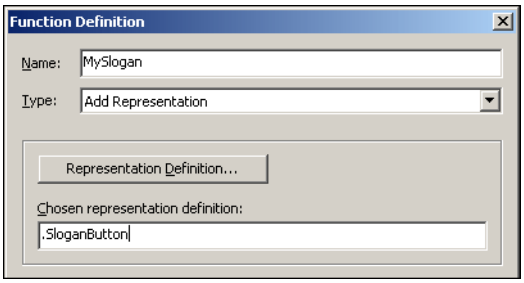
Add the company slogan to the GUI window.

For this example, a new Representation Definition must be written. New Representation Definitions are written and stored in the KnowledgeBase, added to the *Definitions Palette* and to all Representation Definition lists.

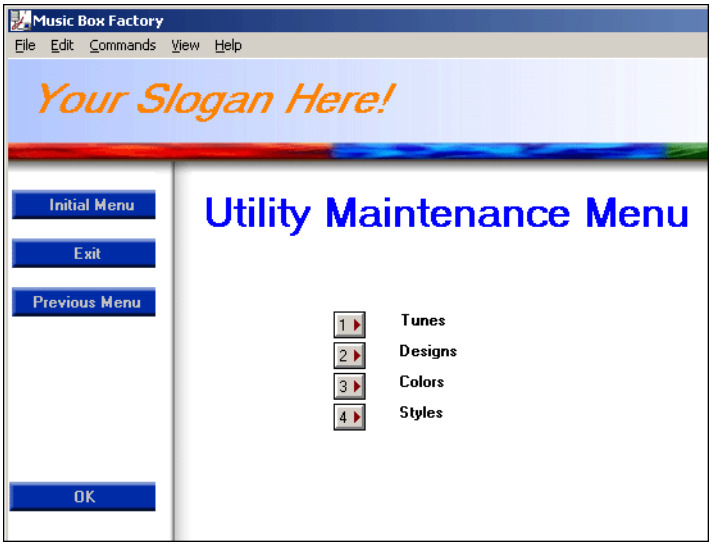
The GUI window before running the Add Representation function definition:



The Function Definition is written after creating the ***SloganButton*** Representation Definition.



The GUI window after running the Function Definition:



Apply Window Layout

Use the Apply Window Layout type to create a Function Definition that applies a Window Layout to the selected controls.

The Apply Window Layout type Function Definition can also be used to nest Window Layouts.

To create a Function Definition that applies a Window Layout:

1. In the *Function Definitions* manager click *New*. The *Function Definition* dialog box opens.
2. Select the *Apply Window Layout* type Function Definition from the *Type* combo box.
3. Select the appropriate Window Layout from the *Chosen window layout* list.

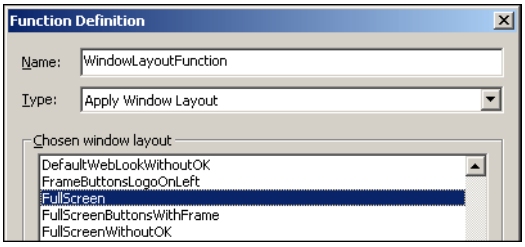


Figure 61. Chosen window layout list

4 Enter a Function Definition name in the name field.

5 Click *OK*. The *Function Definition* dialog box closes. The new Function Definition appears in the *Function Definitions* manager.

Nesting Window Layouts

A Window Layout type Function Definition may be used to nest Window Layouts inside each other. The subordinate Window Layout, after having been defined as a Function, is paired with a Selection Definition within another (primary) Window Layout. When applying the primary Window Layout, the subordinate Window Layout is only applied to the controls selected by the Selection Definition paired with the Window Layout type Function Definition

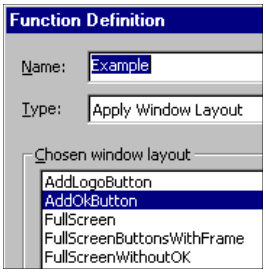
Example 21. Nested window layouts

The *AddOkButton* Window Layout adds, formats, and places an *OK* button. The *AddLogoButton* Window Layout adds and places a logo. In this example, the *AddOkButton* Window Layout is nested in the *AddLogoButton* Window Layout. When the *AddLogoButton* Window Layout is run, the selections and functions contained the *AddOkButton* Window Layout are also applied.

6 In the *Function Definitions* manager and click *New*. The *Function Definition* dialog box opens.

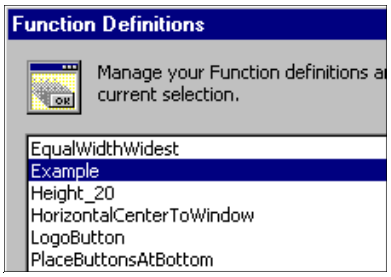
7 Select the Apply Window Layout type Function Definition from the *Type* combo box.

8 Select the Window Layout *AddOKButton* from the *Chosen window layout* list.



9 Enter a Function Definition name in the *Name* field.

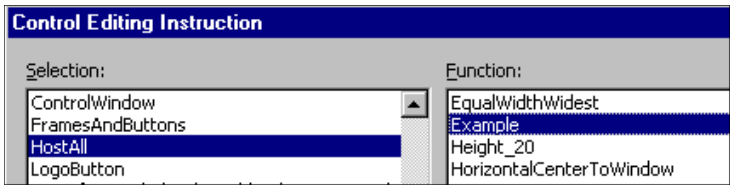
10 Click *OK*. The *Function Definition* dialog box closes. The *Window Layouts Definitions* manager is displayed.



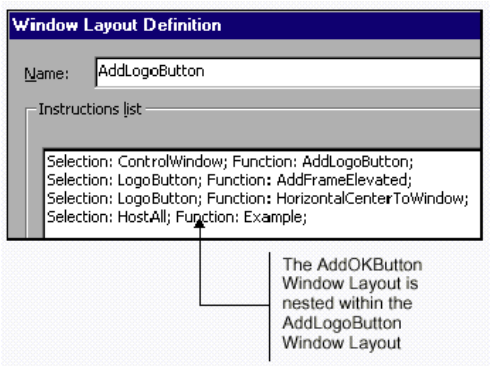
11 Close the *Function Definitions* manager. From the *Design* menu choose *Window Layouts*. The *Window Layouts Definitions*manager opens.

12 To modify the *AddLogo* Window Layout by adding the Apply Window Layout type Function Definition that was just created, from the *Window Layouts Definition* manager, select *AddLogo* and click *Modify*. The *Window Layouts Definition* dialog box opens.

13 In the *Window Layouts Definition* dialog box press *Add*. The *Control Editing Instructions* dialog box opens. Highlight *Host All* for the Selection and *Example* for the Function.



Click *OK*. The instruction line is added to the *Instructions List* in the Window Layout Definition dialog box.



Group of Functions

The Group of Functions type Function Definition assigns multiple commands to a control or group of controls. The Group of Functions type combines multiple Function Definitions under a single definition that can then be applied to selected controls.

Figure 62 shows the sections that appear in the Group of Functions type Function Definition:

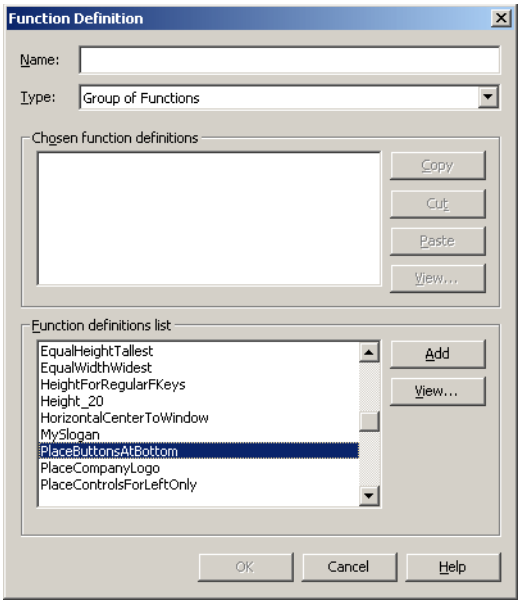


Figure 62. Group of functions

Two list boxes are displayed: The *Chosen function definitions* list box and the *Function definitions list* box.

The *Chosen function definitions* list section displays the Function Definitions added from the *Function definitions list* that will comprise the Group type Function Definition.

The *Function definitions list* section displays all the available Function Definitions. When a definition is selected in the *Function definitions list* the *Add* and *View* buttons are enabled:

- Use the *Add* button to add the highlighted Function Definition to the *Chosen function definitions* list.
- Use the *View* button to view the highlighted Function Definition.

When a Group of Functions type Function Definition is applied to a selection, the functions are applied according to the order in which they appear in the *Chosen function definitions* list. The list can be re-arranged using the editing options that are enabled when a definition is highlighted in the *Chosen function definitions* section.

Table 25. Chosen function definition section options

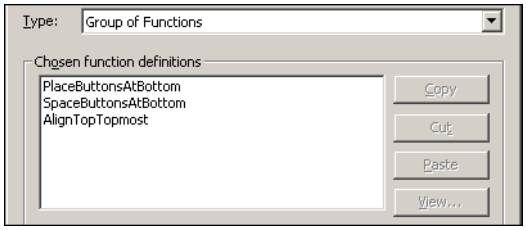
Option	Description
Cut	Cuts the highlighted definition from the list and places it on the clipboard.
Copy	Copies the selection and places it on the clipboard.
Paste	Pastes the clipboard contents into the list above the highlighted definition. If no definition is highlighted, the instruction is pasted at the end of the list.
View	Opens *Function Definition* dialog box. The highlighted function can be viewed or modified.

Example 22. Group of functions

Create a Group of Functions type Function Definition to space and align keys in the command key section using the Function Definitions *PlaceButtonsAtBottom* and *SpaceButtonsAtBottom*.

To create a Group of Functions type Function Definition

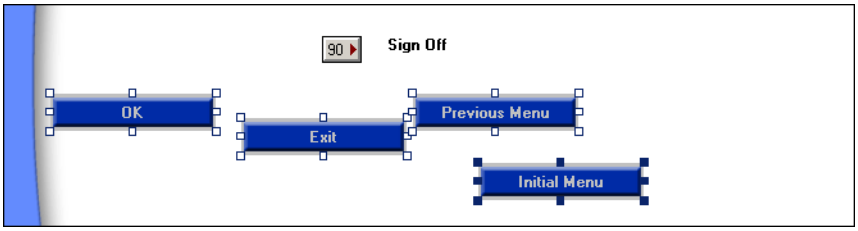
1. Open the *Function Definition* manager.
2. Select *Group of Functions* in the *Type* combo box.
3. Highlight *PlaceButtonsAtBottom* and press *Add*. The function is added to the *Chosen function definitions* list.
4. Highlight *SpaceButtonsAtBottom* and press *Add*. The function is added to the *Chosen function definitions* list.
5. Highlight *AlignTopTopmost* and press *Add*. The function is added to the *Chosen function definitions* list.



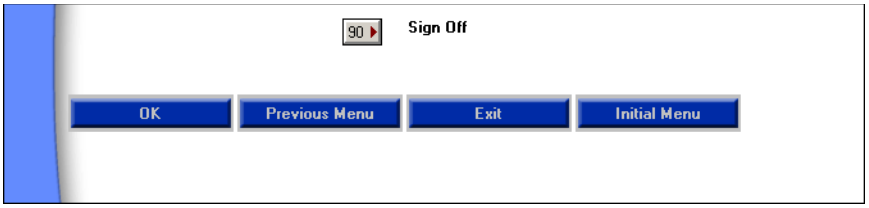
6 Enter a name for this Function Definition in the *Name* field.

7 Press *OK*.

The GUI before running the Function Definition:



The GUI after running the Function Definition:



Chapter 8. Dynamic Sections

The Dynamic Section feature enables you to define a host screen area that can be used to analyze similar host screens on the fly. Dynamic sections have two distinguishable features. First, they are composed entirely of dynamic Pattern Definitions (Dynamic Iterations or Groups). Second, the dynamic Pattern Definitions written to recognize host screen patterns do not require the characters within those patterns to remain static. Non static features of a screen may be input/output fields or Fkeys. When you encounter screens such as these, a dynamic section should be dragged over the variable area(s) of the screen image.

Dynamic sections must contain the entire combination of Dynamic Iterations or Groups found on the host screens such that all host screens will be successfully analyzed. In the runtime Application, screens containing the dynamic areas will be analyzed on-the-fly.

- The screen image used for the analysis must contain at least one character sequence satisfying each dynamic iteration in the dynamic group. If not, then that dynamic iteration will fail to recognize any character sequences in runtime. The control generated in ACE from the sequence is the prototype control.
- All the controls generated in runtime from a given iteration have the same style parameters as the prototype control generated in ACE.

This section includes:

- How Dynamic Sections Work
- Applying Dynamic Sections
- Defining Dynamic Sections in Layout View

How Dynamic Sections Work

The following is an example of the way the dynamic section feature works. In this example we will use two screens from an iSeries with similar layouts. In ACE we will define the input/output area as the dynamic area.

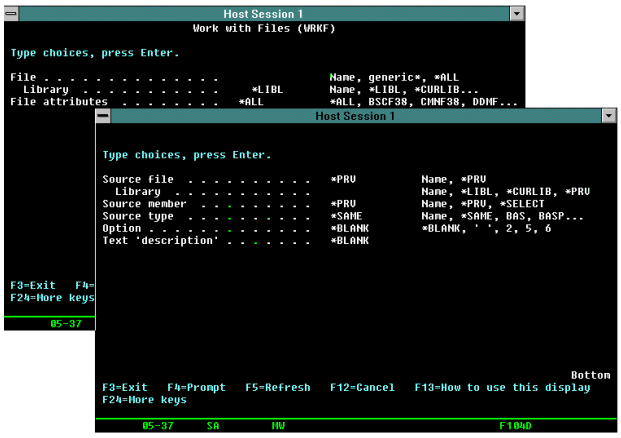


Figure 63. Host screens with dynamic patterns

Notice that both screens contain text with periods, Input/Output fields, and options. The dynamic section will contain Dynamic Iterations that recognize these characters.

Even though the combination of results sent to ACE from the host in a dynamic section can vary, the Dynamic Iterations defining the dynamic sections take care of the conversion. For example, in the sample screens above you see several Fkeys located along the bottom of the screen. You as the developer cannot know during runtime which Fkeys are going to be used. Depending on the particular situation the host may send all, some, or none of the Fkeys to ACE for conversion into GUI.

Fortunately, as long as ACE correctly analyzes “all” the host Fkeys in Analysis View, the screen will be converted properly. You will find that dynamic sections are reliable as long as the screens are analyzed properly.

Applying Dynamic Sections

To apply dynamic sections you must either create a new Subapplication or open an existing Subapplication. Because the example shown here was not done with your exact Application, you may not find the Dynamic Layout from this example in your *Apply Screen Layout* dialog box.

Defining Dynamic Sections in Layout View

To define a dynamic section in layout view:

1. In Layout View select *Layout* > *Change Layout*.
2. You are asked whether to clear the current layout. Click *Yes*.

   The *Apply Screen Layout* dialog box opens.

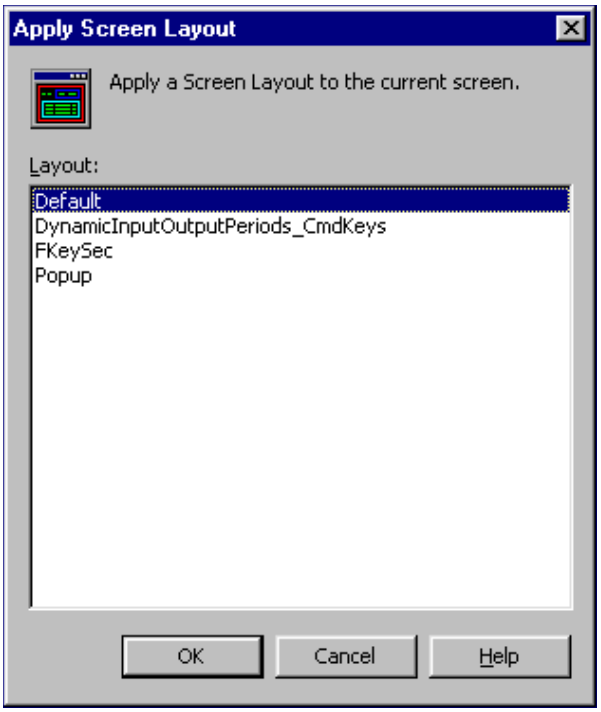


Figure 64. Apply Screen Layout dialog box

3 Select the layout *DynamicInputOutputPeriods_CmdKeys*. Drag the section properly over the screen area. The dynamic area is the input/output area as illustrated in Figure 65.

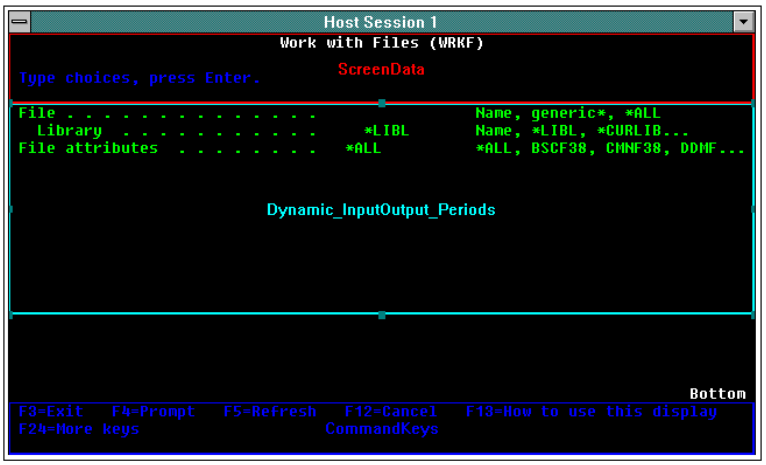


Figure 65. Dynamic Input/Output Periods section

The Pattern Definition for this Dynamic area is as follows:

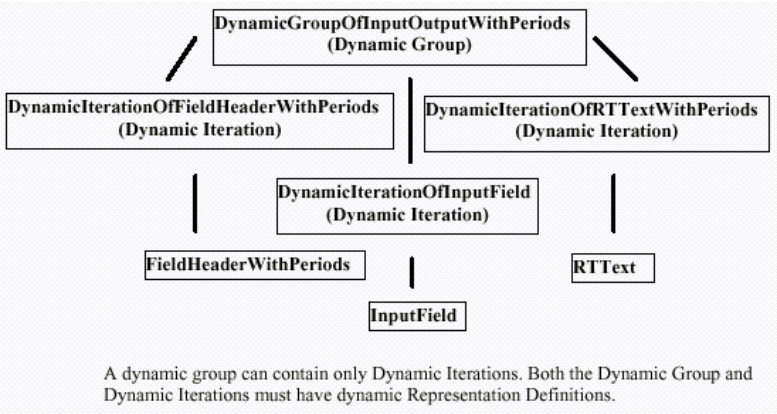


Figure 66. Pattern definition for a dynamic area

Dynamic Areas in Runtime

When you run your generated application with a live emulator, the areas that you have defined as the Dynamic Sections contain the data from the host screen with a GUI representation.

> **Note:** If you have not used the richest complement of Dynamic Iterations, screens with patterns that cannot be converted will only be partially displayed. If problems occur check to ensure that the Dynamic Iterations are comprehensive.

In runtime, ACE searches the dynamic area of the host screen for all the occurrences of the first dynamic iteration. After the first dynamic iteration is identified, ACE searches the remaining parts of the dynamic area for all the occurrences of the second dynamic iteration. This process is repeated for each dynamic iteration.

Hints on Ordering Your Dynamic Iterations in the KnowledgeBase and for Improving Runtime Performance

- Always list the most unique dynamic iterations or least occurring iterations in your dynamic group first.
- The Pattern Definitions for edit fields must precede static fields.
- Use the least number of dynamic iterations as possible. The definitions within the iterations should be simple and have only one representation.

Controlling the GUI Objects from the *.ini File

Changing the values in your `<ApplName>.ini` file can modify the defaults of the GUI objects, which are displayed in the dynamic area during runtime. The automatic adjustments will work in runtime. See the explanation in the "Entries in <ApplName>.ini" on page 286.

Using Dynamic Areas with DDS and SDF Formats

Dynamic areas can be applied to Subapplications originating from DDS and SDF file formats. SDF, or Screen Definition File, is a generic term used to describe a group of mainframe file formats that store information that defines and describes host application screens. To convert such files into a GUI employing dynamic areas, you must edit the DDS or SDF.ini file information. Editing is required because of the way ACE handles dynamic areas.

For example, ACE cannot process dynamic areas that contain output fields with strings of 'O's. Strings such as these provide no match between output fields and Pattern Definitions. Therefore, you must edit the appropriate *.ini file, replacing the unrecognizable string with one that matches a Pattern Definition. After you edit and save the *.ini file, perform screen maintenance on the Subapplication. The new string appears in the field and now the Subapplication can be processed in runtime.

Note: For instructions on editing the DDS.ini or SDF.ini, see "Using DDS.ini and SDF.ini" on page 283.

Part IV. About Enhancing Your Application's Usability

The topics in this section show you ways to improve the usability of the host application without making any changes to the original host application. The issues covered here actually change the way in which the host application performs. Using the features described in this section will make your ACE application easier to use.

The first chapter, Tab Controls, shows you how to add tabs to a window. Tabs can be used to group certain fields that belong together. Just as in Windows, tabs can be viewed only one at a time. ACE also gives you control as to which tab is placed on top as default.

The second chapter, Many-to-One, is another feature that can be used to make your converted application more efficient and easier to use. Many-to-One allows you to combine several host screens into one GUI screen. Using this feature, you redesign original host screens without changing the host application's code. This option is useful when the host application has many screens with similar functions. Using a linking process, ACE combines these screens into one functional GUI.

The third chapter, One-to-Many, is just the opposite of Many-to-One. Using One-to-Many you can take one host screen and create one or more GUI subwindows for it. Dividing a host screen into several subwindows helps organize busy host screens making the ACE application easier to use. Your subwindows can be configured to display in different ways.

The fourth chapter, Skipping Window-Display in Runtime, describes how to improve the performance time of the converted application by using the Skipping Windows feature.

The last chapter, Using Just-in-Time GUI Subapplications, describes special Just-in-Time GUI applications.

Chapter 9. Tab Controls

The Tab control feature enables you to place one or more tabs on a Subapplication window. When a Subapplication window is loaded with many different fields and controls that can be subdivided in related categories, it is a good idea to create tabs.

Placing tabs on a window is a way to divide the information present into several "layers" of data visible one at a time. The Subapplication's window is better organized and looks better, and the user's work is simplified.

This chapter describes:

- Creating Tabs
- Rearranging the Tabs
- Changing Tab Style Options
- Moving Controls from One Tab to the Other

Creating Tabs for Subapplications

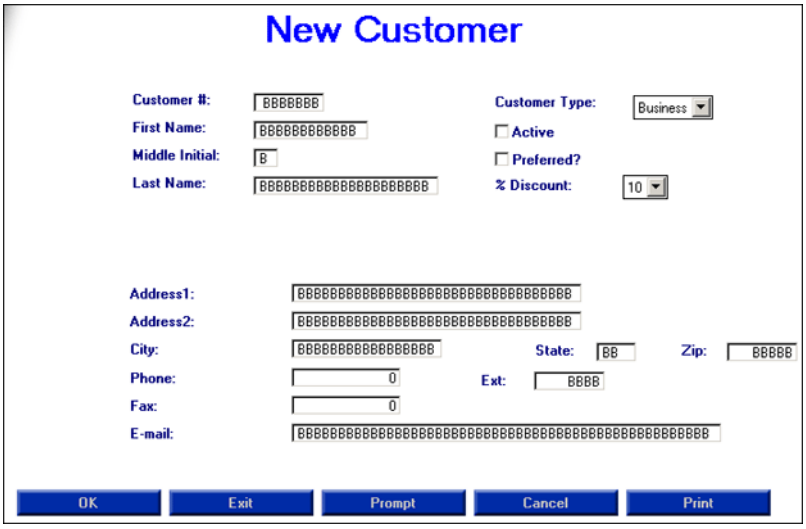
The *Create Tab* wizard is used to create tabs on windows. Tabs are useful in situations where a crowded screen contains data that can be grouped into categories, organizing the visual information to create a less busy window.

The following instructions use an example that shows how information contained in a Subapplication that has many different fields and controls can be divided among three tabs.

Starting the Create Tab Wizard

The first step in creating tabs is to start the wizard:

1 Create a new Subapplication or open an existing one.



This Subapplication displays information about new customers. We can divide the information in three kinds: Name, Phone and Address, and Other Info. Thus we want to create three tabs with these titles.

2 Go to Design View.

3 From the *Design* menu select *Add Control*. The *Add Control* dialog box appears.

4 Select *Tabs* from the *Control Type* selection box, and click *OK*. The *Create Tab* wizard appears.

or

In Design View, click the *Add Tabs* icon in the *Definitions Palette* . The *Create Tab* wizard appears.

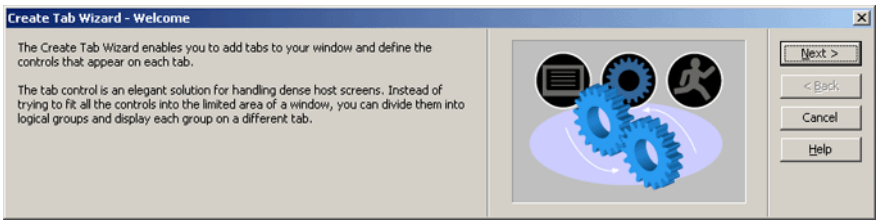


Figure 67. Create Tab wizard

5 Click *Next* to begin creating the tabs.

Creating Tabs

For each tab you create, you must supply the wizard with two pieces of information:

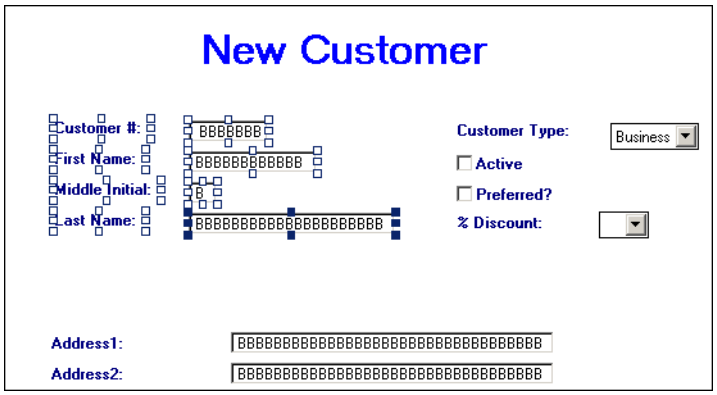
- The name of the tab.
- The controls included on the tab.

You then click the *Add Additional Tab* button and repeat the process for all tabs you want to add.

To create tabs:

1 The first request for information is a name for the first tab. Type the tab's name. In our example, we type: "Name". Click *Next*.

2 Select the controls you want to group on the tab. In our example, the first four edit fields and their descriptive headings.

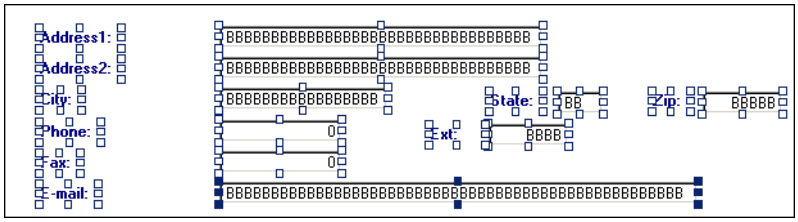


3 Click *Add Aditional Tab* . Several things happen at this time:

- The selected controls disappear from the Subapplication's window.
- The wizard creates the first tab that includes the controls you just selected. You cannot see the tab at this stage of the procedure. You will see it only at the end, once all the tabs have been created.
- The wizard advances to the *Create New Tab* step.

4 Type the name for the second tab. In our example: "Phone and Address". Click *Next*.

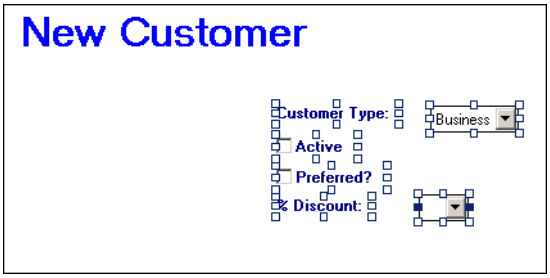
5 Select the controls you want to group on the second tab. In our example, all controls and descriptive headings situated between "Address 1" and "E-Mail".



6 Click *Add Aditional Tab*. The selected controls disappear from the Subapplication's window, the second tab is created, and the wizard advances to the *Create a New tab* step.

7 Type the name for the next tab. In our example: "Other Info". Click *Next*.

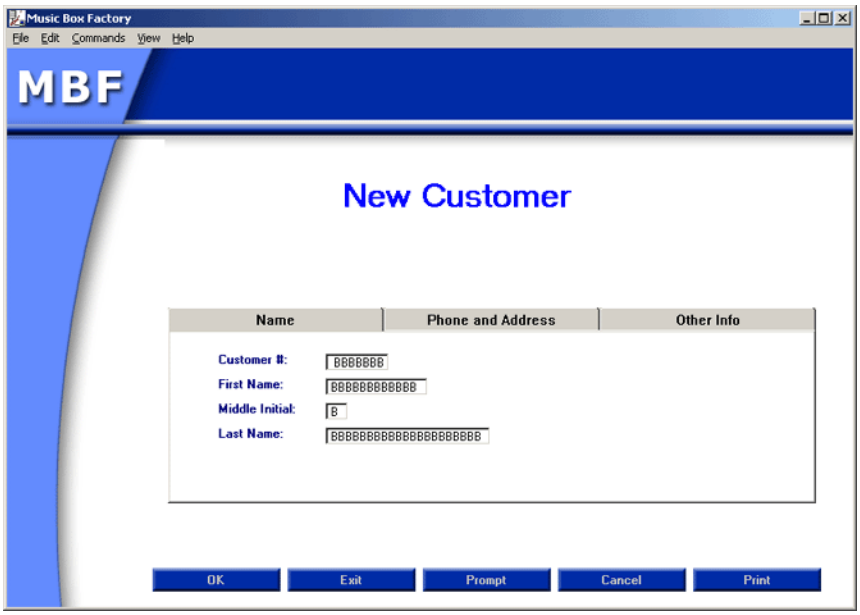
8 Select the controls you want to group on the tab. In our example, the controls and descriptive headings between "Customer type" and "%Discount".



Completing the Wizard

Once all the tabs are defined, you complete the wizard's procedure and arrange the Tab control on the Subapplication window.

1. Click *Next*. The *Specify Top Tab* wizard step is displayed.
2. From the *Top Tab* list, select the tab that should appear on top during runtime.
3. Click *Next*, then click *Finish*. The tabs appear on your window. This is how the Subapplication in the example looks after the Tab control is created.



Rearranging the Tabs

You must now place the tabs in their correct position on the window, and you may also want to modify the position or properties of the controls on each tab individually.

To change the position or size of the tabs:

1 In Design View, click on the tab's caption area. This selects the Tab control.
2 Drag the tabs to where you want them to appear on the window.
3 To resize the tabs, select them and use the cursor on one of the handles.

To modify the position or properties of controls on a tab:

1 In Test View, click on the tab you want to modify.
2 In Design View, select a control or group of controls and move them to where you want them to be on the tab.
3 To modify the properties of a control, double click it to access its style options.

Changing Tab Style Options

To change tab style options, double click on the tab's caption area in Design View. The *Tabs Component* dialog box appears. This dialog box gives you the ability to change the tab that appears on top in runtime. It also allows you to move through the fields using the tab key and to send information back to the host with initial caps.

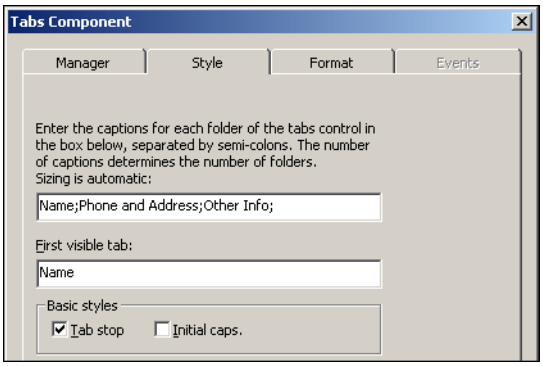


Figure 68. Tab style options

Editing Tabs

Once you have created a Tab control you can perform these editing operations:

- Delete the Tab control.
- Rename one or several tabs.
- Move controls from one tab to the other.

What you cannot do is add, or delete, a single tab to a Tab control.

Deleting a Tab control

You can delete a Tab control you have created. To do this, you first have to remove all the controls on all the individual tabs.

To delete a Tab control:

1. In Design View, delete, or remove from the tabs area, the controls on each of the tabs. This will require you to alternate between Design View and Test View in order to put each of the tabs in focus successively.
2. Press the *Delete* key on your keyboard.

Renaming a Tab

You can rename one or several tabs once you have created them. For this you must change the caption text both in the Subwindow component of the tab and in the Tab component. This is because in ACE individual tabs are in fact subwindows. For more information on subwindows, see Chapter 11 - "One-to-Many" on page 167.

To rename a Tab:

1. In Test View, click the tab you want to rename.
2. In Design View, double click on the tab's body area. The *Subwindow Component* dialog box appears.
3. Change the name in the *Caption text* field. Click *OK*.
4. Double click on the tab's caption area. The *Tab Component* dialog box appears.
5. Change the name of the tab in the caption field. If the tab is the one that must appear on top during runtime, change its name in that field as well. Click *OK*.
6. From the *Design* menu select *Apply Design Changes*.

Moving Controls from One Tab to the Other

To move a control or a group of controls from one tab to another:

1. In Design View, drag the control(s) from one tab to the main window of the Subapplication.
2. Switch to Test View. Select the desired destination tab.
3. Switch back to Design View.
4. Drag the control(s) from the window to the destination tab.

Chapter 10. Many-to-One

Large Applications containing many Subapplications can be difficult to organize, making them hard to work with and maintain. The Many-to-One feature enables combining several host screens and showing them as one window in runtime. Thus, you can reduce the number of Subapplications in your Application, and improve your Application's performance and usability. The Many-to-One feature is especially helpful when Subapplications have a number of fields and controls in common.

Many-to-One works by designating some Subapplications as Dependent and linking them to a Subapplication designated as the Principal. A Principal can have two or more Dependents, while a Dependent can only have one Principal.

This chapter describes:

- The Purpose of Many-to-One
- An Overview of the Many-to-One Workflow
- Creating Dependent Subapplications
- Creating the Principal Subapplication
- Analyzing the Dependents and the Principal
- Representing Dependents as Tabs on the Principal
- Listing Transitions in the <ApplName>.ini file

The Purpose of Many-to-One

Many-to-One allows you to combine several host screens into one GUI screen. Using this feature, you redesign original host screens without changing the host application’s code. This option is useful when the host application has many screens with similar functions. Using a linking process, ACE combines these screens into one functional GUI.

Many-to-One is used to organize several Subapplications so as to display them as one window in runtime. In the window, tabs are often used to organize the information coming from the different screens.

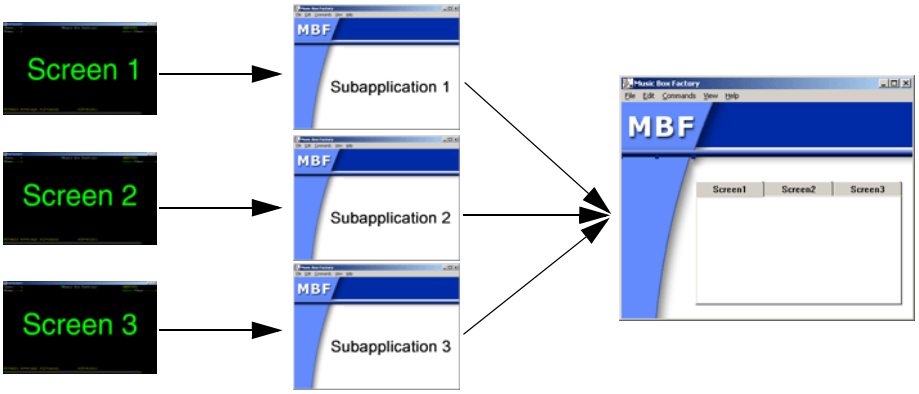


Figure 69. Many-to-One

Many-to-One enables you to re-engineer your application, and improve its usability:

- When there is a sequence of sparse screens that can be combined into a single window. For example, a sequence of logon screens.
- When the information in a number of screens should be displayed together from a functional point of view. For example, employee information - personal, salary, professional qualifications.
- When there are two (or more) related screens on the host. For example, when you move between two related screens using Page Up/ Page Down.

The Many-to-One feature is intended to bring together screens related one to the other either from a functional or from a workflow point of view. We do not recommend “squeezing together” unrelated screens in order to reduce the number of windows in your application.

An Overview of the Many-to-One Workflow

The Many-to-One procedure is based on defining some Subapplications as being of a special type: the Dependent type. You then create an additional Subapplication that is defined as being of another special type: the Principal type. The Principal Subapplication brings together and regulates the interactions between the Dependents. In the Principal Subapplication, the Dependents occur as subwindows which can be transformed into tabs.

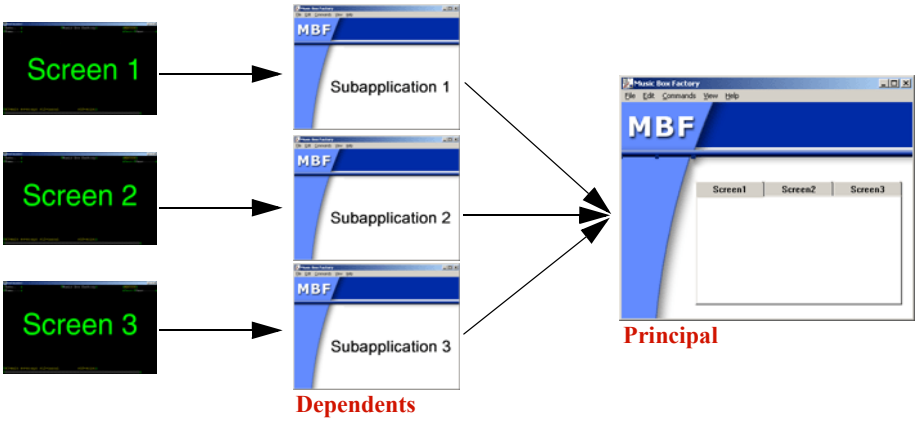


Figure 70. Principal and dependent subapplications

The Many-to-One Workflow

1. Create the Dependent Subapplications before the Principal Subapplication.
2. Create the Principal Subapplication using a "duplicate" screen image taken from one of the Dependents.
3. Fields that are in the same location in all screens are analyzed in the Principal Subapplication. Fields that are unique to a screen are analyzed in the Dependent Subapplication representing that screen.
4. Add toolbar menus and accelerators to the Principal Subapplication.
5. Attach methods in the Principal Subapplication.
6. Add the information that enables the Application to navigate from one screen to the other to the Application's `<ApplName>.ini` file.

Creating Dependent Subapplications

This section shows you how to create a Dependent Subapplication. Designating a Subapplication as Dependent is done in the *New Subapplication* wizard.

To create a Dependent Subapplication:

1. Select the screen to convert using the *New Subapplication* wizard.
2. In the *Subapplication Type* step, select *Dependent*.

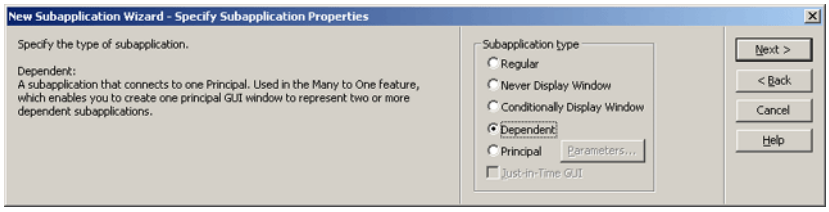


Figure 71. Creating a Dependent subapplication

3 Create the new Dependent Subapplication by completing the *New Subapplication* wizard.
4 Repeat this procedure for each of the Subapplications which will be defined as Dependent.

Note: All Dependent Subapplications must be created before the Principal.

Creating the Principal Subapplication

There is no actual host screen that corresponds to the Principal Subapplication, since the Principal brings together several Dependents. This means that you have to duplicate the screen image of one of the Dependents and use the duplicate to create the Principal Subapplication. There are three options for creating this duplicate screen image:

- Copy one of the Dependent.PNL files and rename it.
- Capture a screen image of one of the Dependent screens.
- If you are working with DDS files, create a new screen image of a Dependent screen using the *Edit Screen Images* wizard.

We will use the first option in the following procedure.

Defining a Subapplication as Principal

These are the steps that are carried out when you create a Principal Subapplication, starting from the .PNL file of one of the Dependents.

To create a Principal Subapplication:

1 Go to the `<InstallDir>\appls\<ApplName>\dds` or the `<InstallDir>\appls\<ApplName>\sdf` folder in your hard drive and make a copy of one of the .PNL files used to make the Dependent Subapplications.
2 Copy the file to another location, preferably not in the ACE folder.
3 Change the name of the file, but keep the .PNL extension.

4 Make a screen image with that file, using the following parameters in the *Create Screen Image* wizard:
   - In the *Screen Image Type* step, select *Captured*.
   - In the *Import Screen Captures* step, select *Import screen captures*.
   - In the *Select Captured Files* step, select the renamed file.
5 Create the Principal Subapplication with the new screen image using the *Create New Subapplication* wizard. In the wizard, select *Captured* as the screen image type.
6 Specify the Subapplication type as *Principal*.

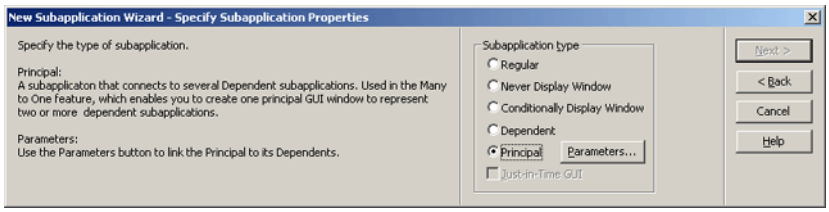


Figure 72. Creating a Principal subapplication

7 Click the *Parameters* button. This opens the *Principal Parameters* dialog box.

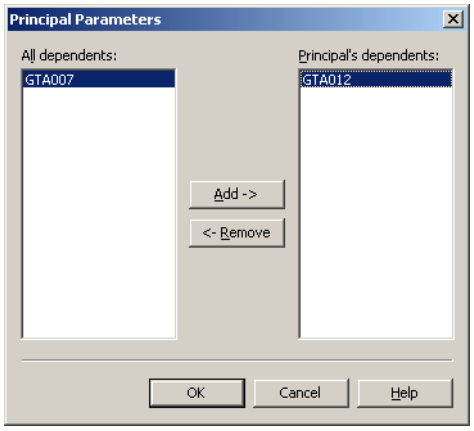


Figure 73. Principal Parameters dialog box

8 The *Principal Parameters* dialog box lists all the Dependent Subapplications which are not yet connected to any Principal in the *All dependents* panel. To link the Dependents to the Principal, select each Dependent from the *All dependents* panel and click *Add*. The Dependents are then transferred to the *Principal's dependents* panel.

> **Note:** Ensure that all the Subapplications which were specified as Dependents are moved to the *Principal's dependents* panel. The process of transferring Dependents from the *All dependents* panel to the *Principal's dependents* panel is the linkage that connects the Dependents to their Principal.

9 Click *OK* and continue the wizard until the Principal Subapplication is created.

Analyzing the Dependents and the Principal

Dependent and Principal Subapplications are analyzed differently. These are the general guidelines for the analysis:

- First analyze the screens of the Dependent Subapplications. Drag an *IgnoreArea* section over fields that have the same position on all the screens. This ensures that the fields represented in the Dependent Subapplications are those that are unique to each Subapplication.
- In the Principal's analysis, do the contrary to what you did for the Dependents' analysis: drag an *IgnoreArea* section over the elements that are unique to each screen. This ensures that the fields represented in the Principal Subapplication are those that are in the same location in each screen.
- Fields that become unnecessary as a result of the Many-to-One procedure, such as FKeys that are used to navigate from one screen to the other, should be covered with an *IgnoreArea* section both in the Dependents and in the Principal.
- The OK button is needed on the Principal Subapplication but not on Dependent Subapplications.
- Dependent Subapplication windows are created without toolbar menus. If you want to add toolbar menus or accelerators, or to attach methods to a control, do so in the Principal only.

Analyzing the Dependents

Start the procedure with the analysis of the Dependents.

To analyze the Dependents:

1 In Layout View, drag an *IgnoreArea* section over the screen elements that are positioned in the same location on all screens.

2 In Layout View, drag an *IgnoreArea* section over fields that will become unnecessary as a result of the Many-to-One procedure.

3 In Design View, apply a ***FullScreenWithoutOK*** Window Layout to the Subapplication in order to eliminate the *OK* button from the subwindow.
4 Repeat the procedure for every Dependent.

Note: Attaching methods to controls is done in the Principal. Do not overlay the WindowCaption section with an IgnoreArea section. Doing so produced Dependents without window captions, which prevents the *Create Tab* wizard from recognizing them and transforming them into tabs.

Analyzing the Principal

To analyze the Principal:

1 In Layout View, drag an *IgnoreArea* section over the screen elements that are unique to each screen.
2 In Layout View, drag an *IgnoreArea* section over fields that will become unnecessary as a result of the Many-to-One procedure.

Example of Dependents and Principal Analysis

Consider as an example the two schematic screens below. This is how they would be analyzed in a Many-to-One procedure in order to create a Principal Subapplication containing the information from both Dependents.

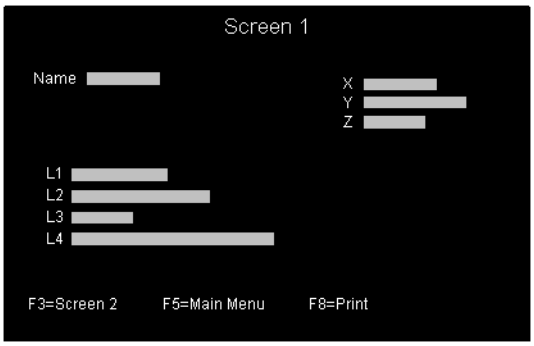


Figure 74. Screen 1

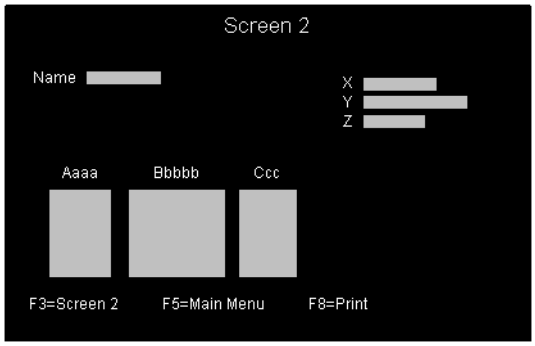


Figure 75. Screen 2

Comparing the two screens shows that:

- The *Name* and *X,Y,Z* fields in the top half are identically located.
- Elements unique to each screen are:
  - In *Screen 1*, the *L1* to *L4* fields.
  - In *Screen 2*, the *List* with the *Aaaa*, *Bbbbbbb*, and *Ccc* headers.
  - The *Fkeys* at the bottom are in identical locations in both screens.
  - The *F3 Fkey* is used in both screens to move to the other screen.

On the basis of this comparison, this is how the screens should be analyzed. The Dependents' analysis is done first.

To analyze the Dependents:

1. In Layout View, drag an *IgnoreArea* section covering the *Name* and *X,Y,Z* fields and another one covering the *Fkeys*.
2. In Design View, apply a *FullScreenWithoutOK* Window Layout to the Subapplication in order to eliminate the *OK* button from the subwindow.
3. Arrange the elements in the window on the smallest possible area, to fit a tab structure.
4. Perform all other necessary local modifications, but remember that you must not attach methods in the Dependents, or create toolbar menus and accelerators.

To analyze the Principal:

1. In Layout View, drag an *IgnoreArea* section over the *L1* to *L4* fields or over the list, depending on which screen was duplicated to create the screen image for the Principal Subapplication.
2. In Layout View, drag an *IgnoreArea* section over the *F3* Fkey. That control is not needed since it is used to move from one screen to the other.
3. Perform all other necessary local modifications. Attach methods to controls if needed. Create toolbar menus and accelerators if needed.

Representing Dependents as Tabs on the Principal

In most Many-to-Ones, the Dependents can be displayed as tabs on the Principal. The ACE Tab control is an elegant solution for handling the information coming from several host screens. This can be accomplished as follows:

> **Note:** Before representing Principal and Dependents as tabs, both the Principal and Dependents must be defined as Subapplications and converted, as described in the previous sections.

To represent the Dependents as tabs on the Principal:

1. Open the Principal Subapplication.
2. Put the Principal's window in focus.

> **Note:** When opening the Principal Subapplication, the Dependent windows and the Principal's window open at the same time. You may be working on a Dependent's window by mistake. Make sure you work on the Principal's window.

3. In Design View, change the caption text of the window to a name different from the name of any one of the Dependent windows.
4. In Design View, select *Design* > *Add Control*. Choose the *Tabs* option and click *OK*.
5. The *Create Tab* wizard appears. Click *Next*.
6. Set the *Create tabs for subwindows* check box. Click *Next*.

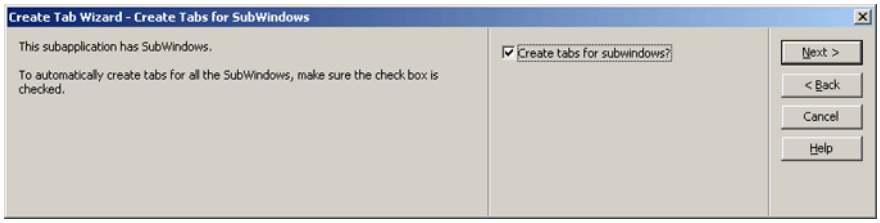


Figure 76. Create tabs for subwindows check box

7. Indicate which tab should be on top. Click *Finish* to create the tabs.

> **Note:** If you skip step 3 you will have problems at the tabs creation step.

Example 23. Displaying two dependent subapplications as tabs

If we display the two Dependent Subapplications used as an example in the preceding section as tabs, we obtain the following window:

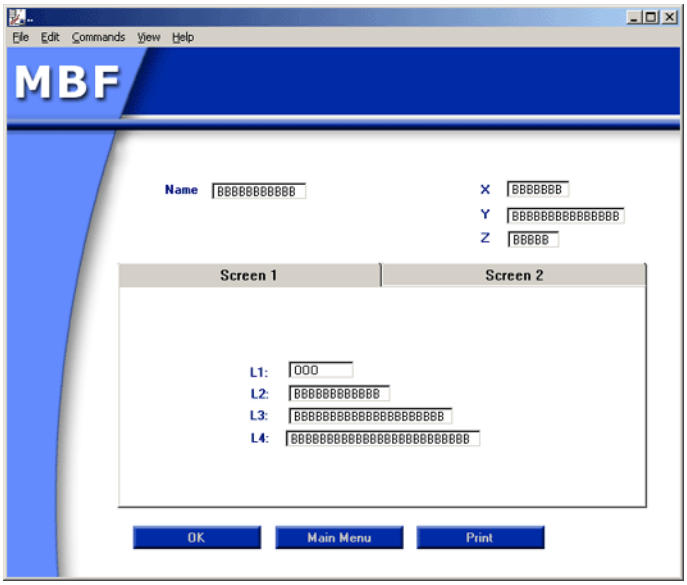


Figure 77. Displaying two dependent subapplications as tabs

Notice that the unique elements appear on the tabs, while the identical elements appear on the rest of the window. Notice also that the *F3* Fkey, used to move from one screen to the other, is not represented.

Listing Transitions in the <ApplName>.ini file

The runtime needs to know how to navigate between the screens of a Many-to-One. This section shows you how to provide that information in the `<ApplName>.ini` file. It also explains when and how you need to use the ***ValidateDependentScreenByName*** DoMethod to ensure the host moves to a specific screen when activation of a method requires this.

Runtime Operation of a Many-to-One

How does a Many-to-One work in runtime? When a regular Subapplication is encountered, the runtime fetches the information from its associated host screen. When the runtime first encounters **any** of the Dependent Subapplications that are

part of a Many-to-One, it displays the associated Principal Subapplication. The Principal must display the information from **all** the Dependents. In order to do that, the runtime must reach each one of the screens and fetch that information.

Two methods are used to navigate between host screens:

- ***UserMoveToDependentScreen*** is activated when a Dependent is reached on the host and ACE displays the corresponding Principal. It makes use of information that you write in the `<ApplName>.ini` file to move from one host screen to another.
- The DoMethod ***ValidateDependentScreenByName*** is used whenever there is a need for the host to be on a specific screen when a method is triggered in the Principal. Using the Dependent's name as a parameter, ***ValidateDependentScreenByName*** verifies whether the host is on the desired screen, and moves the host to the correct screen if needed.

Transition Information

Transition information is the list of host actions that must be performed to move from one screen to another. This information is used by the ***UserMoveToDependentScreen*** method.

In most cases there is only one way to move from one screen to another. For example, to move from Screen 1 to Screen 2 press *F3*, to move from Screen 2 to Screen 3 press *F3* again, to move from Screen 2 to Screen 1 press *Enter*, etc.

Transition Modes

Sometimes there is more than one way to move between two screens. For example, to move from Screen 1 to Screen 2 and update the host you need to type *Enter*, but to make the same move without updating you need to press the *F3* key.

There are three different transition modes:

Table 26. Three transition modes (Sheet 1 of 2)

Mode	Description
Read	Display a window for the first time. Updating data from the screen to the window. This is the default transition mode.
Write	Updating data from the window to the screen. Used when a method contains an "**update**" instruction.

Table 26. Three transition modes (Sheet 2 of 2)

Mode	Description
No Update	Moving to a screen without a request for read or write. Used internally by ***ValidateDependentScreenByName***.

The UserMoveToDependentScreen Method

The runtime uses this method to move between screens when a Dependent Subapplication is first reached. Depending on the screen that is reached on the host, the ***UserMoveToDependentScreen*** method performs the actions necessary to reach the Principal's appropriate Dependents. ***UserMoveToDependentScreen*** is a System-Triggered method that takes values from the `<ApplName>.ini` file as its parameters.

Note: You usually do not modify ***UserMoveToDependentScreen***. Should you want to modify it, do not to use the ***MoveAccordingToHost*** DoMethod line. This could lead to unpredictable results.

Writing Transition Information in <ApplName>.ini Using Host Keys

The ***UserMoveToDependentScreen*** method uses transition information to enable the runtime to navigate between the Dependent's screens.

To write the transition information:

1. Open the `<ApplName>.ini` file, located under the RT32 directory.
2. Add a section called `[PrincipalName_Transitions]`, where `PrincipalName` refers to the actual name of the Principal Subapplication.

The syntax for the new section is as follows:

```
[PrincipalName_Transitions]
Dependents=ListOfDependents
FromDependent_ToDependent_[TransitionMode]=ListOfHostActions
```

The line following the section name gives a list of Dependents, *separated by one blank exactly.*

The following lines list the transition modes and the host actions used to pass from one Dependent to another. Write a separate line for every possible move between Dependents.

In every line following the first, `ListOfHostActions` is a string listing the different host actions that must be performed in order to move from the first to the second Dependent.

Valid host actions are:

Table 27. Valid host actions

Host Action	String
Press a key on the host.	AID (iSeries) key name
Press a key on the host.	\S followed by a key code (as in a *SendASpecificKey* parameter)
Type a string on the host.	\T followed by a string To indicate a blank character in the string, write \B. To indicate several blank characters in a row, type as many times \B as there are blank characters.
Type the value of a global variable at the current cursor location.	\G followed by the name of a global variable.
Move the cursor to a specific location on the host.	\M followed by a four (4) digit number. The first two digits represent the row, the last two digits represent the column. Both row and column numbers are zero based. For example, use 0079 for top right, 2300 for bottom left and 0000 for top left.

Note: The Principal and Dependent names must be written in UPPER CASE letters. The names of the Dependents in the list on the first line should be separated by ONE BLANK space. There should be no blank spaces immediately before or after the equal sign. Inclusion of the transition modes in the line is only required when there is more than one transition mode between two specific screens. This situation is encountered very rarely.

Example 24. Transition section in the <ApplName>.ini file

```
[PrincipalName_Transitions]
Dependents=S1 S2 S3
S1_S2=Enter
S2_S3=Enter
S3_S2=F3
S2_S1=F3
```

The following diagram illustrates a Principal, its three Dependents and the keys that have to be pressed on the host in order to move from one Dependent to another.

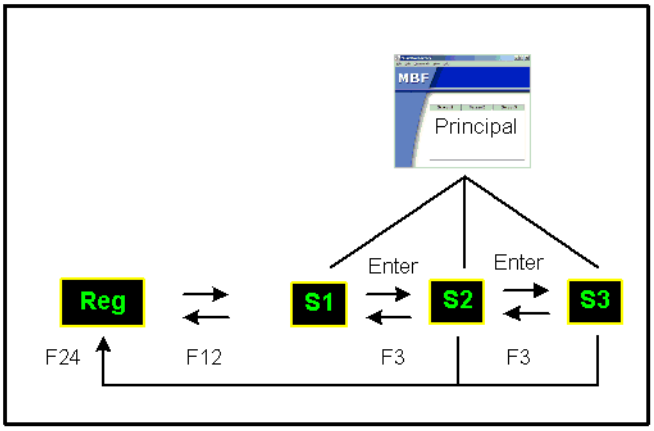


Figure 78. A principal and three dependants

Depending on the screen that is reached on the host, the ***UserMoveToDependentScreen*** method performs the actions necessary to reach the other Dependents.

For example, if Dependent 1 is reached first, Dependent 2 can be reached by pressing *Enter*, and then Dependent 3 can be reached by pressing *Enter* again.

Or, if Dependent 2 is reached first, Dependent 3 can be reached by pressing *Enter*, and Dependent 1 can be reached by pressing *F3*.

For such a case, the transition section in the `<ApplName>.ini` file looks like this:

```
[PrincipalName_Transitions]
Dependents=S1 S2 S3
S1_S2=\EEnter1to2
S2_S3=\EEnter2to3
S3_S2=\EF3From3to2
S2_S1=\EF3From2to3
S1_S3=\EEnter1to2 \EEnter2to3
S3_S1=\EF3From3to2 \EF3From2to3
```

Example 25. How to write transitions when there is a succession of actions

Table 28 shows an example of how to write transition lines when there is a succession of actions to perform on the host in order to move from Dependent 1 to Dependent 2.

Table 28. How to write transitions when there is a succession of actions

Host Transition (S1_S2)	<ApplName>. ini line
F3 “2” Enter	`S1_S2=F3 \T2 Enter`
F5 Home “1 1” Enter	`S1_S2=F5 \S1236 \T1 \B \T1 Enter`
Move cursor to location 0402, type value of variable ABC, Enter	`S1_S2=\M0402 \GABC Enter`

Example 26. When transition modes must be included

Figure 79 shows how the transition section in the `<ApplName>.ini` file is written in the rare case transition modes must be included.

The following diagram shows a Principal and its Dependents, the transition rules that govern the movement from one Dependent to another, and the way the transition section in the `<ApplName>.ini` file is written:

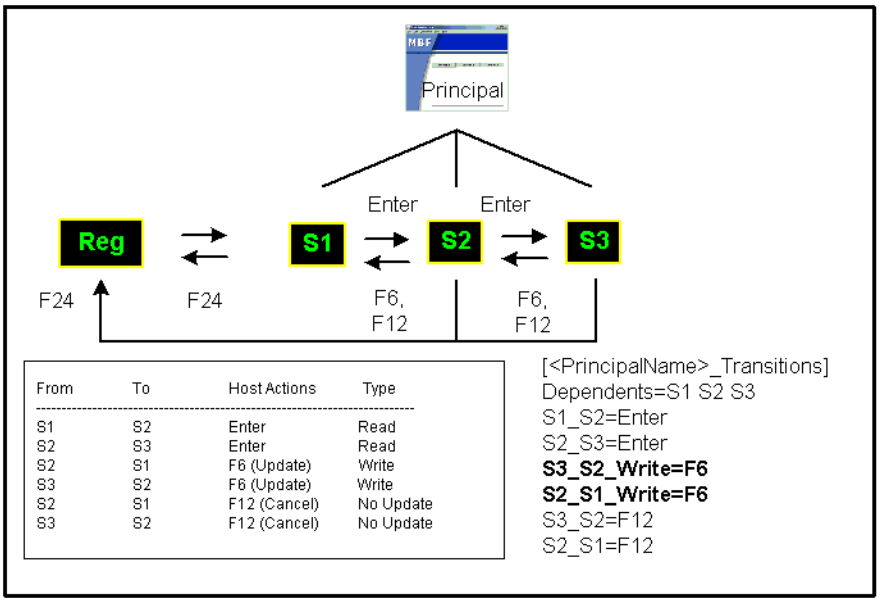


Figure 79. When transition modes must be included

Note that the default transition mode, Read, is not explicitly mentioned in the transition section, since it is the only way to move from S1 to S2 and S2 to S3. For the move from S3 to S2 and S2 to S1, the No Update transition type is not mentioned in the transition section. The transition type that needs to be mentioned is the Write transition type.

The ValidateDependentScreenByName DoMethod

Use the ***ValidateDependentScreenByName*** DoMethod to reach a specific Dependent. You must use this DoMethod whenever a specific action must be performed on a specific Dependent. This DoMethod verifies whether the current host screen is the desired Dependent, and moves to it if it is not.

When the runtime is on the Principal Subapplication, the host is on *one* of the Dependent screens. When you move from tab to tab in the Principal, *the host screen does not move in parallel*, but stays on the Dependent that was first reached. This might cause a problem when a method requiring the host to be on a specific screen is called. In order for the host to be on the correct screen when such a method is called, ***ValidateDependentScreenByName*** must be added to the methods attached to triggers that appear on a single Dependent, and to the methods attached to any controls that require the host to be on a specific Dependent.

> **Note:** The *ValidateDependentScreenByName* method must be attached in the Principal Subapplication only.

Example 27. ValidateDependentScreenByName DoMethod

Let's assume that you must press *Enter* only in S3. You can click the *OK* button on the Principal Subapplication, but the host might be on any one of the Dependent screens, not necessarily screen S3. Therefore, it is necessary to make sure the host is on screen S3 before pressing *Enter*. This is done by inserting the ***ValidateDependentScreenByName*** DoMethod in the ***Enter*** method attached to the *OK* button on the Principal window. The method uses the name of the Subapplication as a parameter.

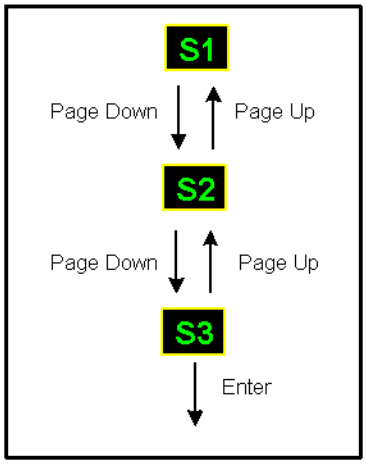


Figure 80. ValidateDependentScreenByName DoMethod

ValidateDependentScreenByName to Modify Enter Method

The ***ValidateDependentScreenByName*** DoMethod is inserted after the method line that updates the screen. Note that it comes before the DoMethod that sets the cursor position according to the focused control, since that operation must be done on a specific screen.

```
Update: Fields: (_All_Fields_) From: TheWindow To: TheScreen
DoMethod: Receiver: `Subapplication` Method: ValidateDependentScreenByName
Parms: (`S3`)
```

```
DoMethod: Receiver: `Subapplication` Method:
SetCursorPosOnScreenAccordingToFocusedControl Parms: ( )
HostType: AidKey: AidEnter RemainInScreen: False
DoMethod: Receiver: `Subapplication` Method: MoveAccordingToHost Parms: ( )
```

Note: You may want to add an If statement after the ***ValidateDependentScreenByName*** method line to make sure the right Dependent screen was indeed reached.

Message Handling

Message lines must be specified in each Dependent (they cannot be specified in the Principal even if they appear on the same line in all Dependents).

Host messages encountered during transitions between one screen to the other are shown on the GUI.

You can define a setting in the `<ApplName>.ini` file that causes the interruption of the transition when a message is encountered.

Set the following in the `[Program]` section:

```
StopDependentTransitionsOnHostMessages=1
```

Any other setting lets the transition continue.

Important Points and Limitations to Remember

The following is a list of the important points and limitations to remember when performing a Many-to-One:

- Fields that are common to all Dependents and appear in the exact same location should be analyzed on the Principal Subapplication.
- Toolbar menus, accelerators and method attachments must be made in the Principal Subapplication only.
- Dependent and Principal screens should have different window captions.
- Add a ***ValidateDependentScreenByName*** DoMethod line to the methods of FKeys or List Commands triggered in specific Dependents. Do so in the Principal only.
- You cannot add or remove Dependents to or from a Principal once the Principal has been defined.
- If in Design View you perform a Restart on a Dependent Subapplication, you must also restart its Principal in Design View.
- Principal and Dependents should be placed in the same library.

- The Runtime Screen Identification information must be accurate for each Dependent. In the Principal this information is not relevant. Make sure you do not mark any message fields on the Principal.
- Runtime Screen Identification changes can be done to the Dependents without restarting the Principal.
- If you make any other type of changes to a Dependent Subapplication after creating the Principal, you need to Apply Design Changes in the Principal.

Chapter 11. One-to-Many

One-to-Many is an advanced feature enabling you to create several windows out of one host screen. By applying One-to-Many you can design subwindows in ACE and display them during runtime. This is especially helpful for developers who convert Subapplication screens containing many fields. Dividing a Subapplication screen into several windows can improve the functionality of your application. In addition, One-to-Many is particularly useful for developers who prefer to use input/output dialog boxes during runtime.

This chapter describes:

- The Way One-to-Many Works
- Subwindows
- Attaching Methods

Subwindows are menu-less windows added to Subapplications. You determine the available options in the subwindow and what the subwindow will look like. In this chapter you see how to create subwindows both via the *Design* menu and by dragging an item from the *Definitions Palette* onto the window. Read about each method of creating subwindows to determine which will work best for your application.

After you learn how subwindows are created you see how to make them work. In order for a subwindow to have effect during runtime you must attach methods to it. Attaching methods is done in Design View. Work through the example given in this chapter to understand how methods for subwindows are created and attached.

The information on methods in this chapter pertains to subwindows only. If you would like more specific instructions about methods please refer to the several chapters dealing with them in *JIS Interface Server: Basic User's Guide*.

The Way One-to-Many Works

Creating a One-to-Many Subapplication is performed in Design View. First call up an existing Subapplication or create a new one. From the *Design* menu choose *Add Control* or drag a representation from the *Definitions Palette*. Both these options provide you with the flexibility to modify Subapplications by adding a single control or completely new subwindows.

ACE goes even further allowing you to configure your subwindows as either pop-ups or as child windows. Additional configuration options allow you to change the default display of the subwindow in runtime to hidden, displayed as an icon, or on top (popup). Hidden subwindows require you to create a method attached to a trigger in order to activate the subwindow.

After configuring the appearance of subwindows you manually drag elements from the main window into the subwindow or attach new features using the *Control* option.

Subwindows

Subwindows are menu-less windows added to a Subapplication. Both the *Control* option in the *Design* menu and the *Definitions Palette* give you the option to create subwindows. The *Definitions Palette* gives you more control over the type of subwindow you wish to create. This is because you can assign unique representations for the windows in the KnowledgeBase and then choose them from the *Definitions Palette* list. The *Definitions Palette* saves time when you frequently use custom designed subwindows throughout an Application.

You can configure subwindows in several different ways. The default subwindow type is a popup. You determine the number of subwindows and which controls should appear in the subwindows.

Creating a Subwindow Using Add Control

To create a Subwindow using Add Control perform the following:

1 In Design View, select *Design* > *Add Control*.
The *Add Control* dialog box appears:

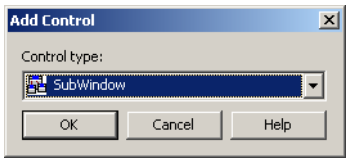


Figure 81. Add Control dialog box

2 Select *Subwindow* from the *Control type* list, and click *OK*. The *SubWindow Component* dialog box opens. Open the *Style* tab:

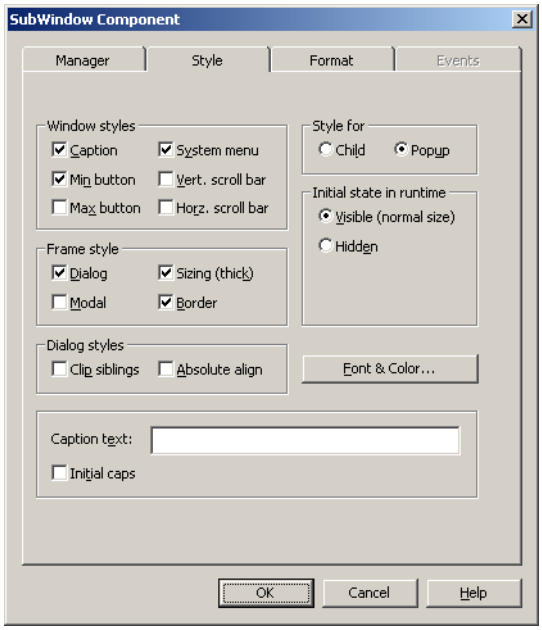


Figure 82. Style tab of the SubWindow component

3 Configure the window style by selecting options from the *Style* tab.

Definitions for the *Style* tab options can be found at the end of this procedure.

4 Click on the *Manager Tab*. Set the *To Window* option in the *Runtime Data Flow* field.

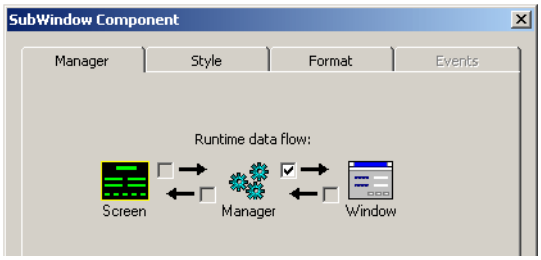


Figure 83. Manager tab of the SubWindow component

Note: If this option is not set the subwindow name will not appear in the *Executed by* list in the *DoMethod:Method Activation* dialog box.

5 Click *OK*. The *SunWindow Component* dialog box closes and a blank subwindow appears in ACE.

6 Transfer controls from the Subapplication to the subwindow using the drag and drop method or assign new controls using *Add Control*. Using *Add Control* requires you to attach methods to the new controls.

The following list explains the features found in the *Style* tab:

Table 29. Window styles (Sheet 1 of 2)

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	When you designate your subwindows as child windows ACE always keeps the subwindow one level above the Subapplication. window.
Popup	A popup window is similar to a dialog box. It has no menu and always appears above the main Subapplication window in runtime. The popup differs from the child window in that it can appear on several different levels on the desktop.
Frame Style	
Dialog	Creates a standard dialog box border.
Sizing (thick)	Creates a standard sizing border that allows the dialog box to be resized.
Modal	Creates a dialog box in which the user is required to click before the application continues. Modal subwindows are commonly used for the notification of an error.

Table 29. Window styles (Sheet 2 of 2)

Window Styles	
Border	Creates a dialog box with a thin border.

Table 30. Dialog styles

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.
Initial Caps	When this check box is set, text from the host has initial capital letters.

Table 31. Initial state in runtime

Initial state in runtime	
Visible (normal size)	The subwindow is displayed on top during runtime.
Hidden	The subwindow is initially hidden during runtime. You need to attach a method and a trigger to the main window that calls up the hidden subwindow.
Font and Color	For information on Font and Color options, see the chapter on Color and Font Design in *JIS Interface Server: Basic User's Guide*.
Caption Text	The text that appears in the title bar of the dialog box/ subwindow.

Creating a Subwindow Using the Definitions Palette

To create a SubWindow using the Definitions Palette:

1. Go to Design View.
2. From the *Definitions Palette* , drag the SubWindow Representation Definition and drop it on the window. A blank subwindow is added at the left corner of the screen.
3. Double-click on the subwindow. The *SubWindow Component* dialog box opens.
4. Configure the window style by selecting options from the *Style* tab.
5. In the *Manager* tab, set the *To Window* option in the *Runtime Data Flow* field.

Note: If this option is not set the subwindow name will not appear in the *Executed by* list in the *DoMethod: Method Activation* dialog box.

6. Click *OK*. The *SubWindow Component* dialog box closes and a blank subwindow appears in ACE
7. Transfer controls from the Subapplication to the subwindow using the drag and drop method or assign new controls using the *Definitions Palette* . Using the *Definitions Palette* requires you to attach methods to the new controls.

Creating a Subwindow Through the KnowledgeBase

To create a Subwindow through the KnowledgeBase:

1. From within the KnowledgeBase window, create a new floating representation, designating it a Subwindow representation component.
2. Save your changes in the KnowledgeBase, and minimize the KnowledgeBase window.
3. Go to Design View. From the *Definitions Palette* select the floating representation you created in Step 1 and drag it onto the window.
4. Double clicking on the new Subwindow opens the *SubWindow Component* dialog box for customization.

Moving Elements into a SubWindow

To move elements into a subwindow:

1. In Design View, select the controls to be moved from the Subapplication window to the subwindow.

> **Note:** Multiple elements can be moved into the subwindow by holding down the shift key and selecting the elements one by one.

2 Transfer the controls to the subwindow using the drop and drag method.

Enabling Display of an Initially Hidden Subwindow

When the initial runtime state of a subwindow is hidden, you need to create a method and associated trigger that activates the hidden subwindow. For the purposes of this example, a button will be used as the trigger.

To add a trigger to a subwindow:

1 In Design View, from the *Definitions Palette* select a Button definition and drag it onto the window.
2 Double click the button. The *Button Component* dialog box appears.
3 In the *Text* field type a name for the button. Click *OK*.
4 Attach a method to the button.

Or

1 In Design View select *Design* > *Add Control*. The Add Control dialog box opens.
2 From the Control type list select *Button* and click *OK*. The *Button Component* dialog box appears.
3 In the *Text* field type a name for the button. Click *OK*. The button appears in the ACE window.
4 Attach a method to the button.

Attaching Methods

It is necessary to create methods in order to hide and show subwindows during runtime. In Design View, you add these methods to the subwindows that were designated as hidden in Design View.

To attach a method:

1 In Design View, double-click the button that triggers the display of the hidden subwindow.
2 The *Button Component* dialog box appears. Click the *Events* tab.

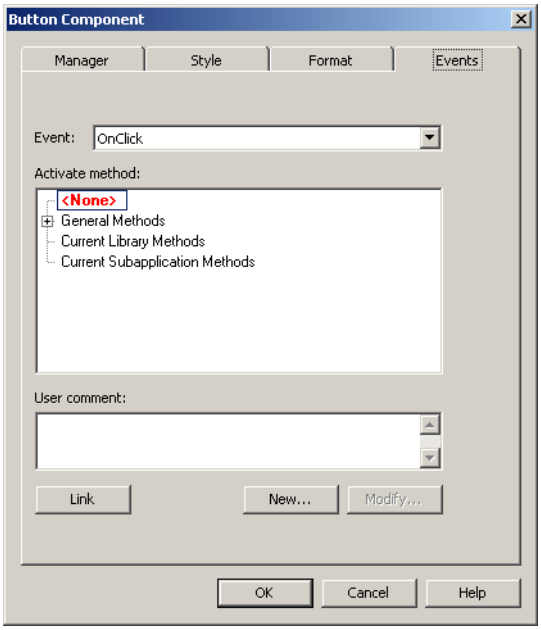


Figure 84. Events tab of the Button component dialog box

3 Click *New* > *Current Subapplication Methods*.

4 The *Define Method* dialog box appears. In the *Name* field enter a name for the new method.

5 Write a method that looks like the method shown in the following dialog box, replacing the '`SubWindow`' receiver with the name of your subwindow.

Note: The variable name of the subwindow can be found in Design View by double clicking on the subwindow. Choose the *Manager* tab and locate the *Name* field in the *Variable* section.

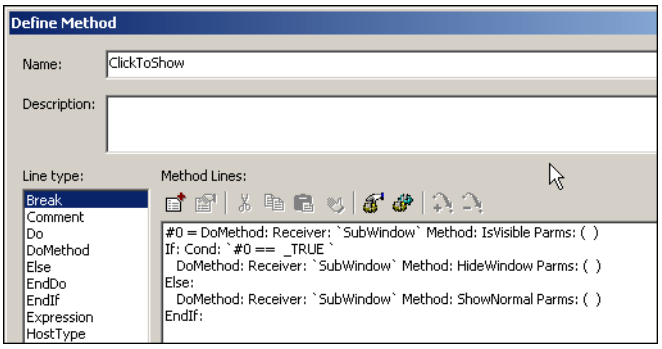


Figure 85. Define Method dialog box

6 Click *OK* twice.

The method is now attached to the trigger. During runtime, pressing the trigger causes the subwindow to open. Pressing the trigger once again causes the subwindow to close.

Chapter 12. Skipping Windows

A Subapplication can be processed in runtime without displaying its window, if information can be received from the host and sent to the host without the end-user's intervention. Essentially the Subapplication's window-display is skipped over.

This chapter describes:

- Skipping Window-Display in Runtime
- Improving Performance Time
- Never Display Window
- Conditionally Display Window
- System-Triggered Methods With Skipping Window–Display
- Auxiliary Methods Used With Skipping Window-Display
- Example Use of the Methods

By utilizing the Skipping Windows feature you can improve the performance time of your converted application. With less information to process and build a GUI, the application works quicker.

You choose the methods that control the skipping process. Windows can be configured to always be skipped or skip only when certain conditions exist. Examples are provided for each situation and the corresponding methods are explained.

It is important to remember that when using this technique the host window is not skipped. Rather, only the GUI equivalent of a host window is skipped.

Skipping Window-Display in Runtime

Skipping the window-display during runtime is governed by two separate factors that you must set in the converter:

- The Subapplication type.
- System-Triggered methods that supply input to the host.

The Subapplication types are: *Never Display Window* and *Conditionally Display Window*. Setting these Subapplication types marks the Subapplication in such a way that the runtime recognizes it as a Subapplication that can be processed without displaying its window. That is, the window-display is skipped.

The System-Triggered methods are: ***UserSkipSubApplication*** and ***UserShouldWindowBeBuilt***. The contents of these methods must contain code that executes operations on the host that the end user usually performs. The methods provide the mechanism for dealing with the host and moving to a new Subapplication.

The advantages of skipping the window-display are two-fold:

1. Improved performance time.
2. Reduced need for the user to perform trivial tasks.

Note: Using this feature requires a solid understanding of methods. Where example methods are given, the method lines appear in **bold** font and the explanation of each line follows in regular font.

Improving Performance Time

During runtime, a regular Subapplication passes through three phases:

- Identifying the screen and building the Subapplication and its variables.
- Building the window and its controls.
- Displaying the window.

Each phase consumes approximately equal amounts of performance time.

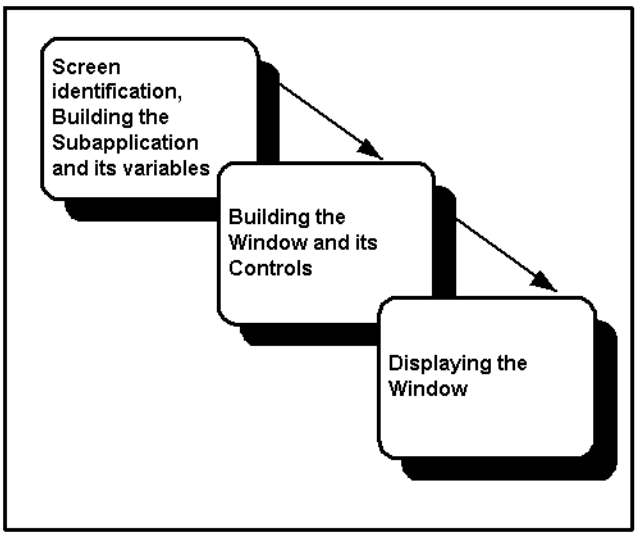


Figure 86. Three phases of a regular subapplication during runtime

The first phase must take place. Each successive phase is optional. This optional use of the last two phases is what allows you to skip the window-display and increase performance. By not displaying the window you save one-third of the

performance time. If you do not build the window and its controls, then by default the window is not displayed and you save an additional one-third of the performance time.

Never Display Window

By setting the Subapplication type as Never Display Window, internal code is modified indicating not to build a window during runtime, the runtime window-display is always skipped, and the Subapplication automatically saves two-thirds performance time. The runtime behaves as if the ***UserShouldWindowBeBuilt*** method returns false. You must write the contents of the ***UserSkipSubApplication*** method for the Subapplication so that the host is supplied with input.

Conditionally Display Window

By setting the Subapplication type as Conditionally Display Window the window-display is skipped in accordance with particular conditions. The conditions are set in the body of the methods, and the decision whether or not to display the window is made during runtime.

The ***UserShouldWindowBeBuilt*** method must be written for this purpose, while ***UserSkipSubApplication*** must be written to provide the runtime with instructions on how to proceed when the window is not displayed.

System-Triggered Methods With Skipping Window–Display

This section describes the system-triggered methods used with skipping window-display.

UserSkipSubApplication

Default: There are no default settings for this method. The method contents must be written by the developer and must contain code that executes operations on the host that the end user usually performs.

- If the return value is TRUE, the Subapplication skips the window-display.
- If the return value is FALSE, the Subapplication's window is displayed prompting the user's intervention.

Customizing Options: The method operates on Subapplications whose Subapplication type was set as either *Never Display Window* or *Conditionally Display Window*. It is activated after the Subapplication and its variables have been built and, when used, after the method ***UserShouldWindowBeBuilt*** has returned a value of either TRUE or FALSE.

For the Subapplication type: Never Display Window:

- The method must contain a method line that instructs the runtime to move to a new Subapplication.
- The return value must be TRUE. This confirms that the window-display will be skipped and the runtime can advance to the next Subapplication.
- If the return value is FALSE a contradiction arises between the method response and the Subapplication type. The runtime fails to proceed to completion and an error message is displayed.

For the Subapplication Type: Conditionally Display Window:

- The return value is dependent on the value returned by the ***UserShouldWindowBeBuilt*** method and on ***UserSkipSubApplication***'s conditions.
- If ***UserShouldWindowBeBuilt*** returns a FALSE value, then ***UserSkipSubApplication*** must return a TRUE value.
- ***UserSkipSubApplication*** must contain a line that instructs the runtime to move to a new Subapplication.
- If ***UserShouldWindowBeBuilt*** returns a TRUE value, then ***UserSkipSubApplication*** can return a value of either TRUE or FALSE.
- If ***UserSkipSubApplication*** returns a TRUE value, the runtime skips the window-display and advances to a new Subapplication.
- If ***UserSkipSubApplication*** returns a FALSE value, the runtime displays the window prompting the user for input.

UserShouldWindowBeBuilt

Location: Manager or General Manager section

Default: Returns TRUE

Customizing Options: This method compliments the ***UserSkipSubApplication*** method. During runtime ***UserShouldWindowBeBuilt*** is activated at every initial entry to a Subapplication whose Subapplication type was set as *Conditionally Display Window*. The method asks whether the runtime should build the window.

- When ***UserShouldWindowBeBuilt*** returns TRUE, the runtime builds the window.
- When ***UserShouldWindowBeBuilt*** returns FALSE, the runtime does not build the window.

Auxiliary Methods Used With Skipping Window-Display

Three DoMethod line types, ***WriteUserVariable***, ***GetUserVariable***, and ***DeleteUserVariable*** are particularly useful when used in conjunction with ***UserSkipSubApplication*** and ***UserShouldWindowBeBuilt***. When used together, ***WriteUserVariable*** and ***GetUserVariable*** let you save variables and their values from one Subapplication to memory, and then use the variables and their corresponding values in other Subapplications. Use the third DoMethod ***DeleteUserVariable*** to remove variables from memory when they are no longer necessary.

Table 32. WriteUserVariable

Method Name	WriteUserVariable
Description	Creates a variable in the variables pool and gives it a value.
Receiver	System
Return type and value	Returns _FALSE in case of failure. Returns _TRUE in case of success.
Parameters	Key- Enter the variable name you want to store in memory. The variable name must be enclosed in quotation marks. Value- Enter the value, or reference a value returned by an existing method line.

Table 33. GetUserVariable (Sheet 1 of 2)

Method Name	GetUserVariable
Description	Returns a variable's value from the variables pool.
Receiver	System
Return type and value	When the variable does not exist, returns an empty string.

Table 33. GetUserVariable (Sheet 2 of 2)

Method Name	GetUserVariable
Parameters	Key- Enter the variable name to retrieve from memory. The variable name must be enclosed in quotation marks.

Table 34. DeleteUserVariable

Method Name	DeleteUserVariable
Description	Deletes a variable from the variables pool.
Receiver	System
Return type and value	Returns _FALSE in case of failure. Returns _TRUE in case of success.
Parameters	Key- Enter the variable name to delete from memory. The variable name must be enclosed in quotation marks.

Example Use of the Methods

Most applications contain a number of menu screens whose main purpose is to direct you to the next appropriate screen as you maneuver through the application. Initially such screens light your way in an unfamiliar environment. As you become more familiar with the application's structure, and the screens that you visit become routine stops, your need for a guiding light is reduced, and instead of being a help these screens may become a hindrance to your work.

Other screens may only require minimal intervention on your part to supply specific values for certain input fields. Some screens contain fields that need to be updated with values from a constant and specific field belonging to another screen. In these cases you can save your end user's time, and increase system performance by skipping the window-display of such screens during runtime.

Never Display the Window in Runtime

The simplest and most straightforward way to skip a Subapplication's runtime window-display is to choose to never display the window. In this case, you must write a System-Triggered method that supplies the input that the host application needs in order to advance to the next Subapplication.

The following example introduces the basic idea that must be followed in order to skip a Subapplication in runtime. The example uses two screens from an Application called the Music Box Factory. After the User Identification screen, the first screen arrived at is the Main Menu. A particular end user always advances directly from the Main Menu to the Distribution Menu.

By designating the Main Menu Subapplication as Never Display Window, the runtime skips the Main Menu GUI, and displays the Distribution Menu, thereby reducing unnecessary and repetitive interaction by the end user with the Application.

Use the System-Triggered method ***UserSkipSubApplication*** to:

1. Place the number 4 in the command menu field.
2. Press the *Enter* key.
3. Instruct the application to move according to the host.

   The runtime displays the Distribution Menu window.

Figure 87. Distribution menu

Setting the Subapplication Type

To set the Subapplication type for New Subapplications:

1 In the *Specify Subapplication Properties* step of the *New Subapplication* wizard select *Never Display Window.*

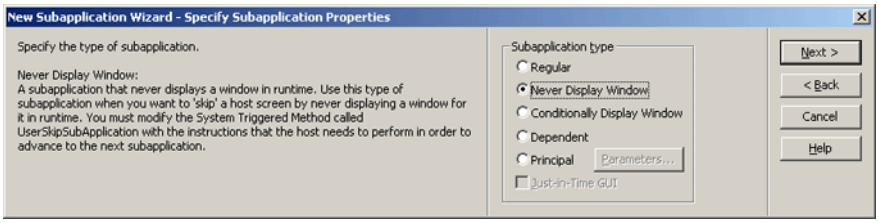


Figure 88. Setting the subapplication type in the New Subapplication wizard

2 Complete the remaining steps in the wizard as required for the Subapplication.

To set the Subapplication type for Existing Subapplications:

1 From the *Subapplication* menu select *Properties*. The *Subapplication Properties* dialog box appears.

2 In the *Subapplication Type* section select *Never* from the *Display Window* combo box.

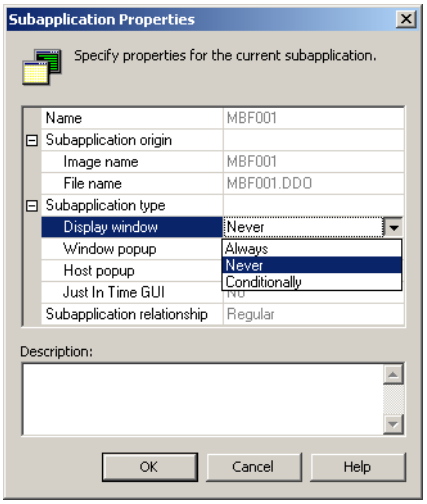


Figure 89. Subapplication Properties dialog box

The UserSkipSubApplication Method

To use the UserSkipSubApplication method:

1 In Design View, select *Design* > *System-Triggered Method*.

2. From the *Current Subapplication methods* list select ***UserSkipSubApplication*** and click *Modify*. The *Define Method* dialog box appears.
3. Enter the method lines to perform the operations on the host that the end user usually performs, and instruct the host to advance to the next screen.

UserSkipSubApplication method lines for this example:

```
HostType: Text: `"4"` Field: MenuCommand MoveCursor: False AidKey: AidEnter
RemainInScreen: False
```

This line types the number 4 in the MenuCommand field, and presses the *Enter* key.

```
DoMethod: Receiver: `SubApplication` Method: MoveAccordingToHost Parms: (  )
```

This DoMethod line instructs the Subapplication to advance to the next screen according to the movement of the host application. As a result, the appropriate GUI window is displayed—in this case the Distribution Menu

```
Return: True
```

Skips the Subapplication's window-display.

Conditionally Display the Window in Runtime

This strategy for skipping a Subapplication's window-display in runtime is used when the window needs to be displayed on some occasions but not on other. Skipping or displaying a Subapplication's window is defined by conditions that are written into the ***UserSkipSubApplication*** method.

In general the condition can take one of two forms:

- Where the condition is host screen independent.
- Where the condition is host screen dependent.

Where the condition is host screen independent you do not need to build the window and you can employ the ***UserShouldWindowBeBuilt*** method to improve performance beyond that achieved by using ***UserSkipSubApplication*** alone.

Where the condition is host screen dependent, if the condition is not met then you need to build the window to prompt the end-user's intervention.

Each form will be presented using a separate example.

Condition is Host Screen Independent

This example expands on the previous one and uses three screens from the application called the Music Box Factory. After the User Identification screen, the first screen arrived at is the Main Menu. A particular end user always advances directly from the Main Menu to the Distribution Menu. However, after working

on Distribution associated tasks the user must return to the Main Menu and navigate to a different module in the application. By designating the Main Menu Subapplication type as Conditionally Display Window, you can write a series of methods that instruct the runtime to skip the Main Menu window when advancing from the User Identification window, and to display the Main Menu window when arriving at it from any other window.

In addition to ***UserSkipSubApplication*** and ***UserShouldWindowBeBuilt***, this example takes advantage of the System-Triggered ***UserInitSubApplication*** method, and the auxiliary methods ***WriteUserVariable*** and ***GetUserVariable***.

UserInitSubApplication

UserInitSubApplication is activated at every initial entry to a Subapplication that is not skipped. This characteristic makes it ideal for determining the name of Subapplications and writing the name to memory. Note that while ***UserSkipSubApplication*** and ***UserShouldWindowBeBuilt*** belong to the Manager section of the Subapplication, ***UserInitSubApplication*** must be assigned to the General Manager section in order for this example to work.

```
#0 = DoMethod: Receiver: `SubApplication` Method: Name Parms: (  )
```

This DoMethod line returns the name of the current Subapplication. In this example that is MBF000.

```
DoMethod: Receiver: `System` Method: WriteUserVariable Parms: ( `#0` ,
`"last SA"` , `""`  )
```

This DoMethod line writes a user variable called **last SA** to memory. The value for the variable is referenced from the first line of the method.

UserShouldWindowBeBuilt

UserShouldWindowBeBuilt is activated at every initial entry to a Subapplication whose Subapplication type is set as either Never Display Window or Conditionally Display Window. ***UserInitSubApplication*** is not activated unless ***UserSkipSubApplication*** returns FALSE and the window must be displayed. In this way there is no clash between their functions.

```
#0 = DoMethod: Receiver: `System` Method: GetUserVariable Parms: ( `"last
SA"` , `""`  )
```

This DoMethod line retrieves the value for the variable *last SA* from memory.

```
If: Cond: `#0=="MBF000"`
```

This If method line references the value returned from the previous line. It is also the condition that determines whether the window should be built. If the variable *last SA* has the value MBF000, then

```
  Return: False
```

Do not build the window

```
Else:
   Return: True
```

Build the window

```
EndIf:
```

This line terminates the condition.

UserSkipSubApplication

```
#0 = DoMethod: Receiver: `SubApplication` Method: UserShouldWindowBeBuilt
Parms: (  )
```

This line calls the System-Triggered method ***UserShouldWindowBeBuilt***.

```
If: Cond: `#0== _FALSE `
```

This line begins the condition. If the window was not built then follow the methods.

```
HostType: Text: `"4"` Field: MenuCommand MoveCursor: False AidKey: AidEnter
RemainInScreen: False
```

This line types the number 4 in the MenuCommand field, and presses the *Enter* key.

```
DoMethod: Receiver: `SubApplication` Method: MoveAccordingToHost Parms: (  )
```

This DoMethod line instructs the Subapplication to advance to the next screen according to the movement of the host application. As a result the appropriate GUI window is displayed—in this case the Distribution Menu.

```
   Return: True
```

Skip the window-display

```
Else:
```

If the window was built

```
   Return: False
```

Display the window

```
EndIf:
```

This line terminates the condition.

Condition is Host Screen Dependent

Using this form, the runtime window-display is dependent on the value of a particular variable that appears in a field on the host screen. This example assumes that a company conducts its workday in shifts. Since each shift performs different tasks on the application, the runtime can be created in such a way that windows which are irrelevant to any one shift are skipped over during that shift. To achieve this, the ***UserSkipSubApplication*** defines a condition that is set

according to the time of day. If one shift works from 06:00 to 14:30 and the next shift starts at 15:00 and continues to 23:30 then the first two digits of the hour on a 24 hour clock can be used to determine if one shift or the other is in progress.

UserSkipSubApplication method lines for this example:

```
Update: Fields: ( Time  ) From: TheScreen To: TheSubApp
```

The Update method line updates the variable *Time* from the screen to the manager.

```
#0 = DoMethod: Receiver: `Time` Method: SubStr Parms: ( `0` , `2`  )
```

This DoMethod line references the value returned in the first line of method. It reads the first two digits of the time and returns them as a string.

```
#1 = DoMethod: Receiver: `#0` Method: AsInt Parms: (  )
```

This DoMethod line references the string returned in the line marked **#0** and converts it into an integer.

```
If: Cond: `#1 >= 15`
```

This line begins the condition. If the integer returned for the line marked **#1** is greater or equal to 15 then

```
HostType: Text: `"9"` Field: MenuCommand MoveCursor: False AidKey: AidEnter
RemainInScreen: False
```

This line types the number 9 in the MenuCommand field, and presses the *Enter* key.

```
DoMethod: Receiver: `SubApplication` Method: MoveAccordingToHost Parms: (  )
```

This DoMethod line instructs the Subapplication to advance to the next screen according to the movement of the host application. As a result the appropriate GUI window is displayed.

```
Return: True
```

Skip the window-display

```
Else:
Return: False
```

Display the window

```
EndIf:
```

This line terminates the condition.

Part V. About Special Host Application Requirements

This section deals with some common application requirements. The topics in this section complement the other ACE features that make the converted application easier to use and more efficient.

The section ties together ACE features that work to finalize the converted application. They are not features central to the conversion process, rather they take care of some loose ends which may have occurred during the conversion.

The first chapter in this section, Using Just-in-Time GUI Subapplications, explains how you can use the JITGUI mechanism to provide a basic GUI in the event that the runtime application fails to recognize a host screen.

The second chapter in this section, Application Specific Configurations, is divided into two topics. First you learn how to deal with fields that extend to more than one line. The second topic concerns the use of the ampersand character in Windows.

The third chapter in this section, Message Handling, explains how ACE handles host messages during runtime and how to configure ACE to display the messages in the GUI. If you have already worked with Pattern Definitions this chapter will seem familiar.

The fourth chapter, Handling Popup Windows, explains what Popup Windows are and how to create them. Creating Popup Windows is easy with the *New Subapplication* wizard. ACE automatically recognizes host screens that are of the popup type and prompts you for the correct configurations.

The fifth chapter in this section, Lists, explains what Lists are and how to work with them. Lists are sequences of vertical columns consisting of one or more items. There are different kinds of lists found in host applications. ACE has Pattern Definitions that recognize lists. ACE also provides a wizard for working with lists in the Creating a Subapplication stage of the conversion process.

The last chapter in this section, External Data, explains how you can supply your application with information from an external data source during runtime, by opening an additional host session from within the running application.

Chapter 13. Using Just-in-Time GUI Subapplications

The Just-in-Time GUI mechanism provides a basic GUI for a host screen in the event that the runtime application fails to recognize it.

This chapter describes:

- Just-in-Time GUI and Unrecognized Screens
- Other Uses of the Just-in-Time Mechanism
- Creating Additional JIT Type Subapplications
- The NO_ATTRS JIT Type Subapplication

Just-in-Time GUI and Unrecognized Screens

In runtime, your application is supposed to identify each transmitted host screen, and then represent it with the corresponding GUI window. The runtime identification works by comparing each transmitted host screen to the list of converted screen images, until it finds the matching Subapplication.

Sometimes, a screen is not identified by the runtime Application. Possible reasons for this are:

- Problems associated with the Runtime Field Information View parameters.
- The screen was modified on the host, but the converted Application was not updated.
- The host screen image was never converted.

ACE has two mechanisms for handling unidentified screens in runtime: the host screen can be allowed to “bleed through”, or it can be represented by a simple GUI, provided by the Just-in-Time GUI (JITGUI) Subapplication. You set which of these ACE uses as a runtime option.

Note that the JITGUI also has a runtime identification mechanism associated with it. However, this mechanism is so general, that it will identify any screen. Consequently, JITGUI is the last Subapplication the runtime compares against a transmitted host screen.

Other Uses of the Just-in-Time Mechanism

The main use of the Just-in-Time (JIT) mechanism is to provide a simple GUI for unidentified screens. The JIT mechanism can also be used to handle screens whose characteristics differ from the "typical" screens of the Application. Such a screen might contain many dynamic variables or fields arranged in an unusual way across the screen. For handling such a screen, you can create additional JIT type Subapplications. The key point when you do this is that the runtime identification mechanism of an added JIT type Subapplication must be general enough to catch all screens with the desired characteristics, but not so general that it catches screens that should be left for the supplied JITGUI Subapplication.

Creating Additional JIT Type Subapplications

To create a JIT type Subapplication:

1. Create a screen image that has the desired characteristics.
2. Start the *New Subapplication* wizard. In the *Specify Subapplication Properties* step, set the *Just-in-Time GUI* check box.

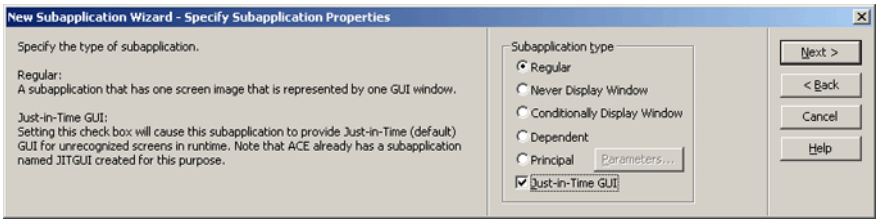


Figure 90. Just-in-time GUI check box in the New Subapplication wizard

3. Create a Pattern Definition to be used as the runtime identification mechanism for the JIT type Subapplication. See the following description of the NO_ATTRS Subapplication for an example.
4. In Runtime Identification View, set this Pattern Definition as the fingerprint.
5. Specify the order in which the runtime Application attempts to identify screens with the JIT type Subapplications. This is set by the order in which JIT type Subapplications are listed in the `JITGUISubapplicationsOrder` line, in the specific.ini's `[Converter]` section.

Example 28. Specific.ini file JITGUI setting

If the Subapplication name is `JIT001`, then the line in specific.ini file should look like this:

```
[Converter]
JITGUISubapplicationsOrder=JIT001;jitgui;
```

The NO_ATTRS JIT Type Subapplication

NO_ATTRS is an additional JIT type Subapplication used for recognizing mainframe screens without attributes. The Pattern Definition, set as the runtime identification fingerprint, is an iteration that covers an entire screen, and whose underlying character set contains precisely the non-attribute characters.

The line in the specific.ini file is:

```
[Converter]
JITGUISubapplicationsOrder=NO_ATTRS;jitgui;
```

The screens that are not identified in runtime are first passed to the NO-ATTRS Subapplication. If the screen does not contain any attribute characters, it will be represented by the NO_ATTRS GUI. If the screen does contain attribute characters, NO_ATTRS fails to recognize it, and the screen is passed to the JITGUI Subapplication.

Chapter 14. Application-Specific Configurations

This chapter covers topics which pertain to applications that have already been converted. Essentially they are features that further customize the way the converted application looks and performs.

This chapter includes:

- Creating Multi-Line Text Boxes
- Support for the Ampersand (&) Character

Creating Multi-Line Text Boxes

The multi-line feature for text boxes allows entering a forced line break within a text box on the GUI. During runtime, you enter a forced line break by pressing *Ctrl+Enter* within the multi-line text box.

The forced line breaks are translated to blank spaces on the host.

Figure 91. Creating multi line text boxes

Multi-line text boxes can only be tested during runtime.

To configure a text box for multi-line functionality:

- Designate the selected text box as multi-line in the *TextBox Component* dialog box in Design View. Scroll bars can be appended to a multi-line text box by setting the *Vertical scroll* and/or *Horizontal scroll* boxes that are enabled when *Multi-line* is set. During runtime, the vertical scroll bar is always visible; the horizontal scroll bar is automatically displayed when needed.
- Write a new, Subapplication-specific, User-Triggered update method for Subapplications that use multi-line Text Boxes and link the new method to a control that initiates returning data to the host.
- In Runtime Field Information View, make space for the number of characters sent to the host.

Designating a Text Box as Multi-line

To designate a text box as a multi-line:

1. In Design View, double-click on the text box. The *TexBox Component* dialog box opens.
2. Go to the *Style* tab and set the *Multi-line* check box.
3. Optional: Set the *Vertical* and/or *Horizontal scroll* check boxes.

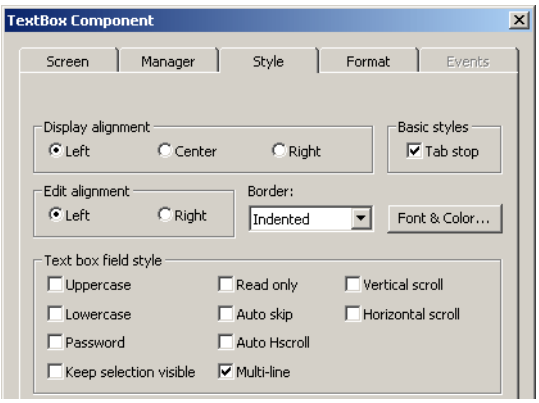


Figure 92. Style tab of text box component

4. Click *OK*.
5. Resize the text box to accommodate more than one line of text.
6. Click the *Apply Design Changes* button.

Writing an Update Method for Multi-line Text Boxes

The default ***Enter*** method must be modified to accommodate multi-line text boxes. The following instructions show the steps for writing this alternate ***Enter*** method.

1. Prepare the following information:

   The multi-line text box component variable name.

   Count the number of characters accepted by the multi-line field on the host.
2. In the *Design* menu, choose *User-Triggered Methods*. The *User-Triggered Methods* dialog box opens.
3. Choose *New* > *Current Subapplications Methods*. The *Define Method* editor opens.
4. In the *Line type* list, double-click *Update*. In the *Update: Update Subapplication Variables* dialog box, enable update from Window to Screen and, under *Variables to update*, choose *All*.
5. Click *OK*. The Update method line is added to the Method Editor.

6 In the *Line type* list, double-click *DoMethod*. In the *DoMethod: Method Activation* dialog box enter the following information:

DoMethod	***AsPaddedLinesString***. This method converts the new line character entered on the GUI into blanks accepted by the host.
Executed by	Select the multi-line text box component.
Assign Values	Assign values for the first line, middle line (all lines except the first line), and the maximum number of characters. The maximum number of characters may not exceed the number of characters counted on the host.

7 Add a HostType method using the multi-line text box variable.

The method that updates the host with data entered to the specified multi-line text box looks like this:

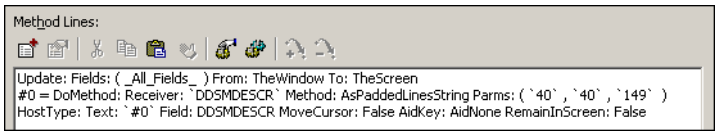


Figure 93. Method that updates host with data in the specified multi-line

8 Link the method to a control. For example, link the new update method to the OK button.

Allowing the Maximum Number of Characters

The space the input field for multi-line text boxes must be configured, in the Runtime Field Information View, to accommodate the maximum number of characters accepted by this field on the host.

Example 29. Allowing the maximum number of characters

The Inscription input field on the host accepts up to 140 characters; 66 characters appear on the first line and 74 characters appear below the first line (the screen shots show only a portion of the field). On analysis, the input field that follows the static control Inscription yields a text box. The 66 characters from the first line appear within the text box.

The input field in Analysis view:

Figure 94. Input field in Analysis view

The input field in Design view:

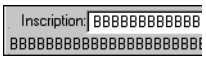


Figure 95. Input field in Design view

In Runtime Field Information View, examining the *Decomposition Definition* dialog box for this field, shows that only the number of characters in the first line are recognized:

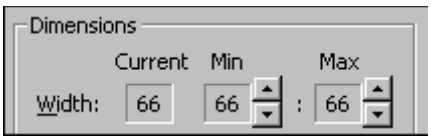


Figure 96. Decomposition Definition Properties dialog box

To allow typing the maximum number of characters on the GUI, the input field property must be set to the maximum number of characters accepted by the host. For this example, the maximum width should be set to 140.

To modify field properties:

1. Go to Runtime Field Information View.
2. Click the input field. Press *F6* to select the lowest level definition in the input field and check *Lock selection* in the *Decomposition Definition Properties* dialog box.
3. In the *Decomposition Definition Properties* dialog box, change the width dimension to reflect the maximum number of characters accepted by input field on the host.

Support for the Ampersand (&) Character

Windows uses the ampersand character (&) as a prefix to indicate that the letter that follows is a first-letter-mnemonic. For example, the letter “S” in the option “Save” is a first-letter-mnemonic. This letter is programmed as such by placing an ampersand character before the letter “S” in the application program.

Chapter 15. Message Handling

A Message Definition is a Pattern Definition that recognizes host screen messages during runtime. Once a message has been recognized, ACE handles the message according to the method that is attached to the Message Definition. The method converts the host screen message into a GUI characteristic which you define.

This chapter describes:

- Message Definition Types
- Messages Definitions View
- Message Display
- Creating Multi-line Message Boxes
- Message Waiting Indicator
- Message Help

Message Definition Types

ACE allows designation of certain Pattern Definitions as Message Definitions. The following Pattern Definition types can be designated as Message Definitions:

- String
- Horizontal Group
- Horizontal Iteration
- OneOf
- Character Set

Messages Definitions View

The Messages Definitions View interface is used to designate existing Pattern Definitions as Message Definitions. Although the interface is similar to Pattern Definition View you cannot create Pattern Definitions here.

To access Message Definitions View, first open the KnowledgeBase and then click the **Message Definitions View** icon as illustrated in Figure 97.

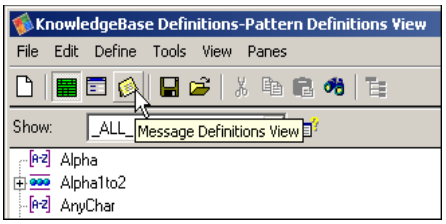


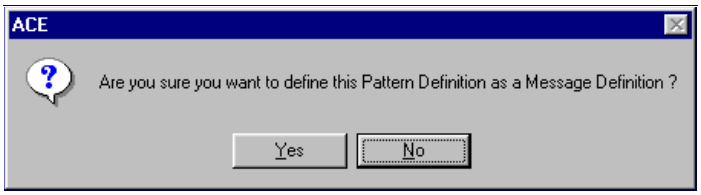
Figure 97. Message Definitions View icon

The upper left **Pattern** pane displays all the Pattern Definitions that exist in your KnowledgeBase. The bottom **Message Definitions** pane displays all the Message Definitions that have already been saved.

Designating a Pattern Definition as a Message Definition

To designate a Pattern Definition as a Message Definition:

1. From within the KnowledgeBase click the **Message Definitions View** icon.
2. Select a Pattern Definition from the **Pattern** pane.
3. From the **Define** menu click **New**. You are asked to confirm. Click **Yes**



   The selected Pattern Definition appears in the Message Definitions pane.
4. Attach a method to the Message Definition.

> **Note:** If you select a definition name that is already defined as a Message Definition, a warning message appears.

Message Handling Methods

Messages often appear during runtime operation. Messages may be sent by the system, the application, or another source. There are various types of messages, such as error messages, informational messages, input messages, etc.

Example 30. Message handling methods

Error message: "`Incorrect Password`"

Notification message: "`Reconnect Successful`"

Input message: "`Enter UserId`"

Message handling methods deal with messages that may appear during the generated Application's runtime. Every time a message appears, the runtime finds and executes the method that handles this message. A message handling method should provide a suitable response to a message, so that the generated Application can continue running.

Defining a Message Handling Method

To define a message handling method go to Design View. From the **Design** menu select **Message Handling**. The **Message Handling** dialog box opens.

Figure 98. Message Handling dialog box

Table 35. Message Handling dialog box options (Sheet 1 of 2)

Option	Description
Message definition name	Contains a list of all the Message Definitions defined in Analysis View.
Message type	Displays the type of the currently selected Message Definition.

Table 35. Message Handling dialog box options (Sheet 2 of 2)

Option	Description
Notification	A Notification message displays information to the user, but it does not require a response from the user. For example, the message "Reconnect Successful" displays information to the user but does not require a response.
Context	Defines the context of the Method—the entire Application, or the Screen only. The context of a message handling method depends upon the message: There are messages that may appear only in a specific screen. For such a message, the context of the message handling method may be limited to the Screen only. There are also messages that may appear at any view of the Application. For such a message, the context of the message handling method should be the entire Application.

To define or modify a message handling method:

1. In Design View, from the **Design** menu select **Message Handling**. The **Message Handling** dialog box appears.
2. From the **Message Definition Name** list, select the name of the Message Definition which will be handled by the method.
3. **Message Type** displays Notification or Input (response required) according to the type of the selected definition.
4. Specify the context of the method by selecting the appropriate button. The **Message Definition Name** list displays all the Message Handling methods that are of the currently selected context. To view the Message Handling methods that are of the other context, change the selected radio button.
5. Click the **Method** button.The **Define Method** dialog box opens.
6. Define or modify the method and click **OK**.

> **Note:** Currently, it is not possible to refer to fields on the screen when defining a Message Handling method. Defining a Message Definition method does not affect what method, if any, is attached to the underlying Pattern Definition in the KnowledgeBase.

7 In the **Message Handling** dialog box, click **Close**. The **Message Handling** dialog box closes and the changes are saved.

Message Display

ACE can be set to handle messages either before or after the window is displayed. Message handling is set in the `[Program]` section of the `<ApplName>.ini` file as follows:

To display the messages before the window is displayed:

```
HandleMsgsAfterRefresh=0
```

To display the messages after the window is refreshed:

```
HandleMsgsAfterRefresh=1
```

Creating Multi-line Message Boxes

Occasionally the iSeries host application displays a multi-line message. On the host system, only one line of the message is displayed at a time. A small "+" sign is displayed at the right hand side of the line indicating that the message has additional lines. In the host application, the PageUp and PageDown keys are used to scroll through the message.

ACE runtime enables you to view multi-line messages from the host screen using a similar mechanism. When a message has more than one line, two small buttons are displayed on the message box. One button displays a “+” sign, the second displays a “-” sign.

To view the second line of the multi-line message, click the “+” button. The message field then scrolls to the next message line. Click the “-” button to scroll upward to the beginning of the message.

> **Note:** The “-” button is disabled on the first line of the message and the "+" button is disabled on the last line of the message.

Setting Up Your System to Handle Multi-line Messages

Perform the following procedure to set up your system for multi-line message handling:

1. Call up the `<ApplName>.ini` file.
2. In the `[Program]` section set `HandleNextPrevMessages=1`

Note: Currently this setup is the same as for Message Waiting Indicator.

Message Waiting Indicator

iSeries users can configure their systems to display a visible indication that a message is waiting. When this is configured, a Message Waiting Indicator (MWI) is displayed instead of having messages burst into the Application while working.

When a message is to be displayed, a small push button with Message Waiting text is displayed on the right hand side of the DIL. The button is visible only when a message is waiting. When the user clicks the MWI, ACE performs a ***FlatMoveTo*** to the message window.

Handling the Message Waiting Indicator

Perform the following procedure to set up your system for the Message Waiting Indicator:

1. Call up the `<ApplName>.ini` file.
2. In the `[Program]` section set `HandleNextPrevMessages=1`

Note: Currently this setup is the same as for Multi-line Message Boxes

Message Help

The iSeries has a help key on the keyboard that can be used in the GUI Application to call up help information about the displayed message from the host application. When a message is displayed the user places the cursor on the message field and presses the **Help** key. If the application contains help for that message, the system displays the related help text.

Enter the instruction to display message help handling in the `[Program]` section of the `<ApplName>.ini` file as follows:

```
HandleMessageHelp=1
```

When this entry is added to the *.ini file, a button displaying a question mark “?” is displayed on the right side of the DIL. When the user clicks on this button, the System-Triggered method ***UserHostMessageHelpRequest*** (receiver is the Subapplication) is activated.

Chapter 16. Handling Popup Windows

A host popup is a screen that opens on top of the current screen in the host application. It is usually used in the host application for selecting a value to be inserted into an input field on the screen. When the cursor is on an input field the user is instructed to press a function key, which prompts a popup screen. The value that the user selects from the popup screen is inserted into the input field in the main screen.

This chapter describes:

- How ACE Identifies a Host Popup Window
- Creating Popup Windows With the Wizard
- Controlling Popup Display in Runtime
- Processing a Popup Window Subapplication

How ACE Identifies a Host Popup Window

ACE identifies a host popup by the unique pattern created by its borders. You will only be able to see these patterns when you view your screen with attributes visible. To make attributes visible, select *Show Attributes* from the *Emulator* pull-down menu. With attributes visible, you can see two distinct columns of attributes. These constitute the borders of the host popup screen. See the screen below.

The Pattern Definitions for the borders have been included in the KnowledgeBase. These Pattern Definitions are not primary, but they have an assigned priority.

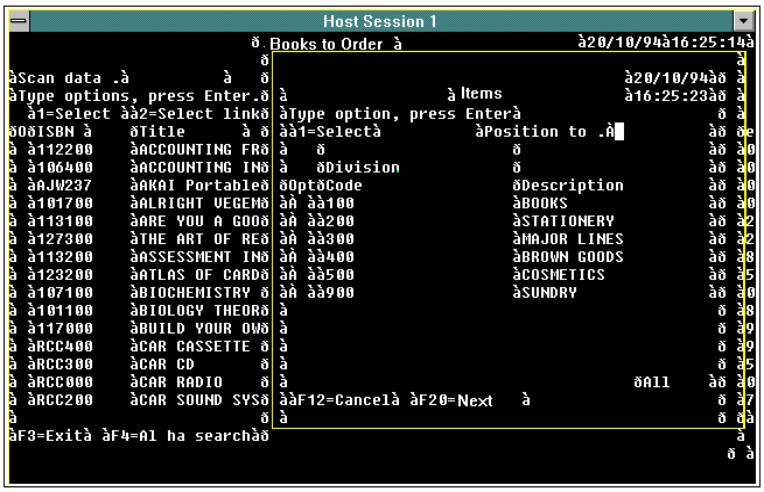


Figure 99. A host popup identified by its unique border

The Pattern Definitions that define the popup border are standard and the *New Subapplication* wizard automatically picks up just about all popup windows. That means that the Pattern Definitions in the KnowledgeBase that identify popup windows will in most cases identify the popup windows in all applications.

ACE identifies host popup screens automatically by looking for the border pattern. ACE starts at the cursor location and looks for the border pattern by searching outward. Therefore ACE can only identify a popup window correctly if the cursor is within the popup window when the screen is captured. Almost all pop-ups have the cursor within the popup window because the host application is generally programmed with the cursor at the point of the next user input. When the popup window is displayed, the next user input is inside the popup.

> **Note:** If the cursor is not in the popup window, do not try to force it manually. This causes a conflict in the runtime identification stage, and causes a failure.

Creating Popup Windows With the Wizard

If the host screen you are working with contains the patterns ACE recognizes as a popup, you will be prompted to give certain information concerning the popup characteristics of the screen.

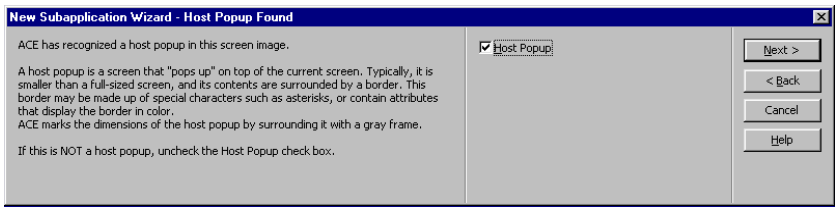


Figure 100. Identifying pop-ups in the New Subapplication window

For example, many pop-ups are of the list variety. In this case the *New Subapplication* wizard requests list information so that the popup and all its contents will be analyzed correctly. The wizard takes care of most of this information automatically, but in certain instances you are requested to provide additional information to complete the conversion process.

Controlling Popup Display in Runtime

ACE enables you to control the way pop-ups are handled when the runtime application is running in host-driven mode. When in host-driven mode, ACE is automatically set to display pop-ups overlaid on the parent screen.

If you prefer not to have the screen overlaid in runtime, in Layout View go to *Subapplication* > *Properties* and in the *Subapplication Properties* dialog box change the *Window popup* field to *No*.

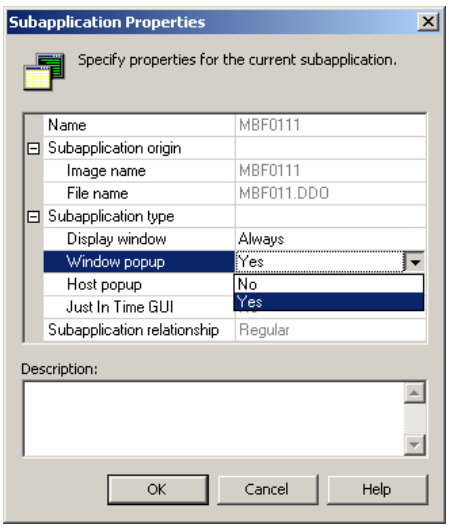


Figure 101. Window popup field in Subapplication Properties dialog box

When set to *No*, the popup is displayed as a full screen in the runtime application, when in host-driven mode.

This feature also enables full application screens that are not host pop-ups to be displayed as pop-ups in the runtime application when in host-driven mode. Set a full screen to be displayed as a popup in the runtime application by either setting the *Popup Window* check box in the *New Subapplication* wizard or through the *Subapplication Properties* dialog box.

Processing a Popup Window Subapplication

When ACE converts screens with popup windows, it relates only to the area of the screen containing the popup. Areas outside the popup window are ignored.

Layouts

Analysis begins by prompting you to select a layout that is appropriate for the popup window. The selected layout is superimposed on the area of the popup window and adjusted proportionately to the popup window size. This layout then assumes the dimensions of the popup window.

Note: If you save a layout that defines a popup window, it will be saved in small format. Be careful not to inadvertently save a system layout, or it will be displayed in compressed format when it is called up to analyze a full screen.

Batch Mode

Because they are identified automatically, there is no need to process pop-ups separately.

Message Handling

Popup windows have fewer lines than the full screen window. Therefore, host messages cannot be displayed automatically, as they are in full screen converted screens. It is necessary to mark message sections manually.

Runtime Screen Identification

In Runtime Screen Identification View, the popup window is marked according to its contents: Fixed, Variable, Message Line, Fingerprint. The Fingerprint should appear inside the popup window area.

Displaying the Popup in Runtime

The converted popup can be displayed in your runtime Application anywhere on the screen. There is no limitation as to the size or location of the converted popup.

Chapter 17. Lists

A list is defined as a sequence of vertical columns consisting of one or more items. The items in a row are separated by spaces and/or attributes and are aligned to the left or to the right, to create columns. A folded list may break this mold by "wrapping around" the column. More information on folded lists is given later on in this chapter.

This chapter describes:

- The List Structure
- Working with the List Wizard
- Recognizing a List
- List Type Pattern Definition Structure
- Placement
- Searched Columns
- Arranging List Columns and Headers Manually
- Modifying the Type of a Column
- Types of Lists
- Processing Lists Through the Views

Lists are arranged as tables in host applications. When creating Subapplications in ACE the *List* wizard takes you through most of the necessary steps used to create the GUI equivalent for a host list.

ACE uses special Pattern Definitions placed in the section of a layout to recognize lists during analysis. ACE also allows you to arrange the look of the list by arranging columns and linking the headers to columns.

Complete instructions are given here that explain how to configure ACE to convert host screen lists into functional GUI designs. You are also instructed on how to customize the look of the GUI and modify the structure of the list.

The List Structure

Lists can generally be broken down into identifiable sections and this is how ACE recognizes them during analysis. The identifiable sections in lists are described in the diagram below.

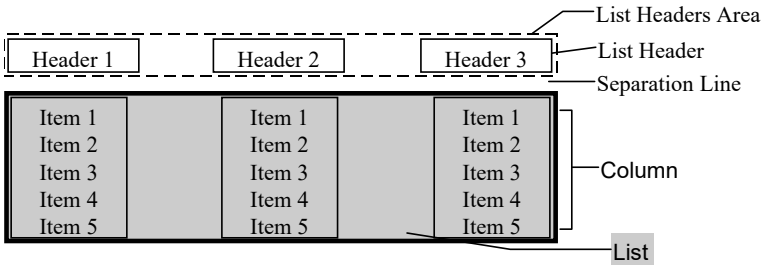


Figure 102. List elements

Table 36. List elements

Option	Description
List Header	Text that describes the contents of a list column. List headers usually appear above the list, and above the columns they describe. The header can be a single line or multi-lined.
List Headers Area	The area above the list where the List Headers can be found.
Separation Lines	Empty lines between the List Headers and the List. The number of separation lines varies between lists, ranging most often from zero to two.

Note: The term “List” is synonymous with the term “Subfile” in iSeries. It is extremely important to identify the list elements correctly. If not, ACE fails to analyze the screen properly.

Example 31. Lists

The following screen contains a list with headers, ten rows, and six columns.

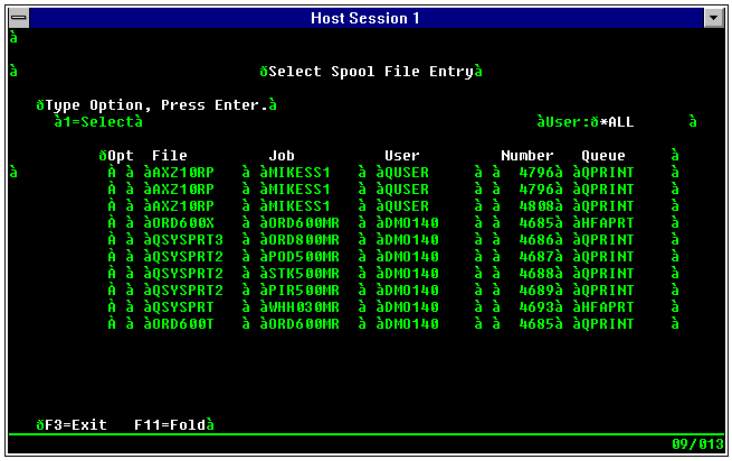


Figure 103. List in host view

Working with the List Wizard

The *New Subapplication* wizard automatically recognizes lists and prompts you for certain information to allow determining a list's type and its delineation. A new layout can be defined in Layout View, with the aid of the *List* wizard, for cases where a list has not been analyzed correctly or where two lists appear in the same Subapplication. This chapter explains lists so that you will better understand the wizards' role in list conversion. Additionally you will be shown how to define lists for yourself in Layout View if the need arises.

Recognizing a List

ACE uses Pattern Definitions placed in the list section of a layout to recognize lists during analysis. The structure of the Pattern Definitions used to identify a list matches the hierarchical structure of a list. Thus a list is composed of columns, and columns are composed of headers and items.

The following schematic drawing illustrates the way ACE analyzes a list.

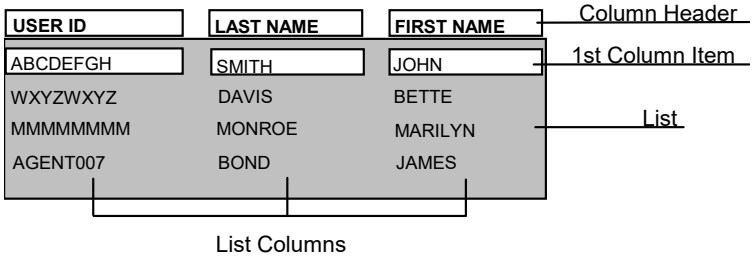


Figure 104. The way ACE analyzes a list

In an analyzed screen, the list area includes all its rows, but not its headers. The Column Header and the first Column Item of each column are analyzed. The first Column Item represents all the column's items. Any modifications to the first Column Item influence the entire column.

> **Note:** While a header accompanies most columns some columns may appear without one.

List Analysis Process

A clear understanding of the list analysis process will provide a solid foundation for your work with lists. The following describes the stages involved in the list analysis process and how analyzed lists appear on the screen.

The list analysis process is executed in three stages; each stage symbolized by a color on the screen.

Finding the List Area

In the first stage of the list analysis process ACE locates the area in the host screen occupied by a list. The list width and height are determined and this area is represented on the screen as a blue rectangle. In conjunction with this the List Headers area and separation lines are established. Even though they are found at the same time it is important to remember that the List Headers area and the separation lines do not form part of the list area.

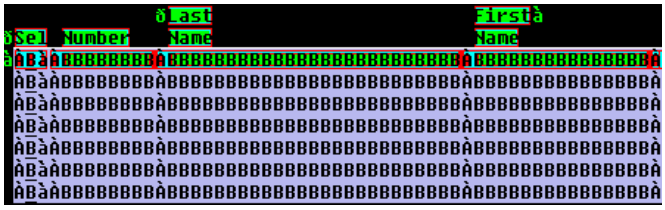


Figure 105. List area

Finding Columns, Headers and Linking Headers to Columns

Two processes are performed in this stage. The first process identifies different columns by their location and width. This information is imposed on the first item of each column in the list and is expressed visually by the color red. Any alterations to the first item in a column will effect the entire column. The second process finds the headers and then tries to link these headers to the appropriate columns. The headers are also marked in red.

You will find it easier to see the results of the processes performed in this stage by viewing the list in View Links mode. In Analysis View, right click on the List and select *View Links* from the shortcut menu.

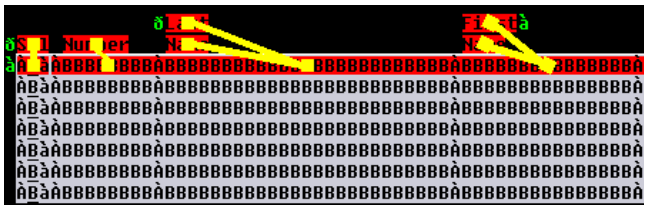


Figure 106. Links between headers and columns

Finding the Header and Column Values

The final stage looks for the values and items that make up the headers and columns. These are the characters that are displayed in the Design View during conversion, and it is these characters to which you attach Representation Definitions. In an analyzed list this information is colored green.

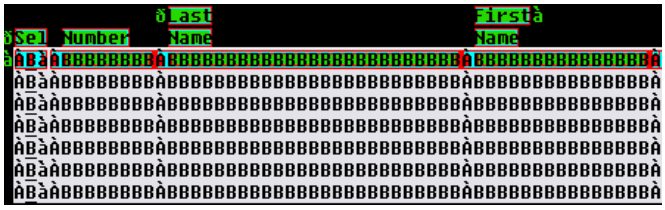


Figure 107. List header and column values

List Type Pattern Definition Structure

This section describes the pattern definition structure of the patterns that are used to analyze lists in ACE.

The Pattern Definition Structure of List with Parameters

A List with Parameters type Pattern Definition, must have precisely six child patterns, and each of these six child patterns plays a specific role within the Pattern Definition. The six child patterns are called the arguments of a List with Parameters. ACE uses the arguments to recognize and analyze lists in the three stage list analysis process described earlier. You edit the arguments of a List with Parameters Pattern Definition in the same way that you edit any other child pattern.

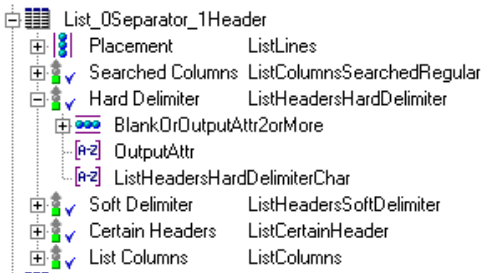


Figure 108. Replacing list arguments

- You can replace an argument with a different Pattern Definition. For example, the *Placement* argument ***ListLines*** could be replaced with the Pattern Definition ***ListNotSubfileLines***.
- You can edit the argument Pattern Definition either in place or as a top level Pattern Definition. For example, you can edit the *Hard Delimiter* argument, which in the above Pattern Definition is ***ListHeadersHardDelimiter***, in the usual way.

You can open the *KnowledgeBase Definitions* window with the correct *List with Parameters* type Pattern Definition displayed for editing in a number of ways. A handy means of opening the *KnowledgeBase Definitions* window is to use the right click shortcut menu in Analysis View.

To edit a List with Parameters type Pattern Definition in the KnowledgeBase:

1 In Analysis View, position the mouse pointer anywhere in the list and right click.

2 Select *Modify* from the shortcut menu. The *KnowledgeBase Definitions* window appears with the correct List with Parameters type Pattern Definition displayed.

You can also double click anywhere inside the area of the analyzed list.

Argument Description

The arguments of a List with Parameters type Pattern Definition are:

Table 37. List arguments

Argument	Description
Placement	The child pattern that determines the screen area that is included in a list.
Searched Columns	The child pattern that identifies the location and size of each list column.
Hard Delimiter	The child pattern that, within a given header row, makes a coarse division of the header row into column headers. ACE analysis does not join characters on different sides of a hard delimiter into a single header.
Soft Delimiter	The child pattern that, within a given header row, makes a fine division of the header row into column headers. ACE analysis *may* join characters on different sides of a soft delimiter into a single header.
Certain Headers	The child patterns, usually strings, that are the character sequences that will always be identified as column headers. This identification overrides ACE analysis with the hard and soft delimiters. The supplied ***CertainHeaders*** pattern is a OneOf type Pattern Definition whose single child pattern is a dummy string. You must replace the dummy string with strings or other Pattern Definitions corresponding to character sequences that you are certain are column headers.
List Columns	The child pattern that identifies the kind of information contained in a column.

Placement

The aim of the *Placement* argument is to determine the screen area that will be included in a list. This area should include all the list items and their attributes. The *Placement* argument is itself defined by the Pattern Definition that is attached to it, and it is therefore the flexibility of this Pattern Definition that is, to a large extent, responsible for determining which items can be included in the list. Other factors affecting the size and position of the screen area included in a list will be discussed later. Let's look briefly at the Pattern Definition that is attached to Placement for most lists of Lists with Parameters type.

ListLines

The Pattern Definition ***ListLines*** is the *Placement* child pattern for most Lists with Parameters. For such list Pattern Definitions, any element that satisfies ***ListLines*** is included in the list area.

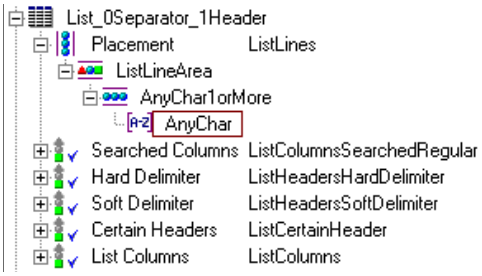


Figure 109. ListLines

If you have a closer look at ***ListLines*** you will see that it is a vertical iteration of ***ListLineArea*** which ultimately is a horizontal iteration of ***AnyChar***.

Two important points must be emphasized that are implied by the nature of ***ListLines*** and by the aim of the *Placement* argument:

- ***ListLines*** is a Pattern Definition that accepts every screen element.
- Some external limitations must be placed on the influence of ***ListLines*** in order to restrict the list area to a particular screen region.

Restricting the Influence of ListLines

As pointed out, left unrestricted, ***ListLines*** would accept nearly every screen element. Its activity is restricted because a List with Parameters Pattern Definition is itself restricted to the area of the section containing it. Thus, the list area recognized by, say, ***List_0Separator_1Header*** is the area of the primary section *List_0Separator_1Header*.

Placement With Screen Captures

Screen captures by default do not contain filter sections. Thus, their analysis is more dependent on the way you handle the section containing the list. In such a case, closely enclose the list, the list attributes, and the list headers with the list section border as indicated in the diagram below. In this way the area matched by ***ListLines*** is restricted to the screen area enclosed by the list section.

> **Note:** When working this way the area enclosed by the list section is equivalent to the placement, and therefore the placement includes the list headers area and separation lines.

Figure 110. Placement with screen captures

Placement Includes List Header

At the bottom of the of the *Parameters* tab for List with Parameters *Pattern Definitions are* two check boxes:

- Placement includes list headers
- Verify SDF columns using Searched Columns definition

The *Placement includes list headers* checkbox determines whether or not the header is located within the list area found by the *Placement* argument.

By default this checkbox is not set and placement is determined according to information given by the DDS file. When analyzing captured panels the *Placement includes list headers* checkbox should be set. ACE then refers to the values provided in the *Parameters* tab for the *Number of header lines*, and the *Number of lines between header and list.*

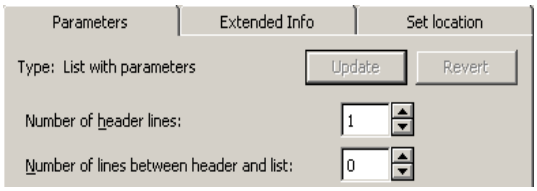


Figure 111. List parameters tab

ACE uses these values to determine how many placement lines to reserve as header lines, and how many lines to leave as separator lines.

Number of Header Lines

The value listed in the spin box determines the number of lines, starting from the top of the list that are reserved for list headers.

Number of Lines Between Headers and List

The value listed in the spin box determines how many separator lines are placed between the last header line and the first list line.

The ACE KnowledgeBase supplies List with Parameters Pattern Definitions with up to three header rows and 0 or 1 separator rows.

Example 32. Number of lines between headers and list

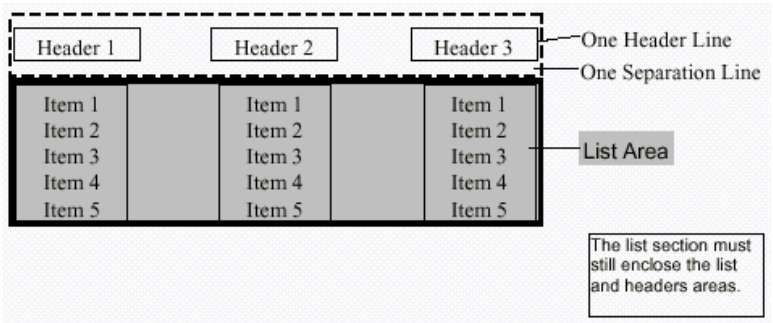


Figure 112. Number of lines between headers and list

Placement With DDS Panels

Placement is defined by a combination of the following:

- List filter
- Placement Pattern Definition.

The List filter provides the boundaries of the list. The Pattern Definition ***ListLines***, lets ACE accept any character inside the list boundary as part of the list. It also defines a minimum number of rows for the list. The minimum number

of rows currently defined is four rows. This minimum helps ACE avoid recognizing two or three lines that happen to be aligned in columns as a list, when in fact they are not a list.

ACE continues to use the Pattern Definition attached to the *Placement* argument, in our case ***ListLines***, to find the list area. However instead of the list section restricting the influence of the Pattern Definition, this function is performed by a combination of the list filter and background filter included in the DDS information. Since the list filter allows a match for ***ListLines***, whereas the background filter does not, the list area and consequently the placement are defined by the extent of the list filter.

The list filter only finds the list area. A separate filter finds the header area, and as a result the placement does not include the headers.

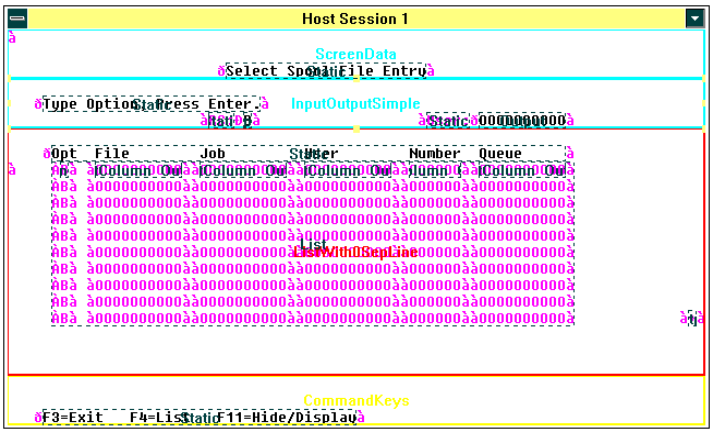


Figure 113. List filters and the placement pattern definition

Searched Columns

ACE uses the *Searched Columns* argument for two purposes:

- In non DDS applications it is used to identify the position and size of each column in a list.
- In DDS applications it is used to determine the nature of columns extracted from the host under rare and specific circumstances.

Non DDS Applications

ACE scans the first record of a list from left to right finding the character sequences that satisfy the Pattern Definition attached to the *Searched Columns* argument.

The location and size of these sequences determine the location and size of the list columns. By default the ***ListColumnsSearchedRegular*** Pattern Definition is attached to the argument.

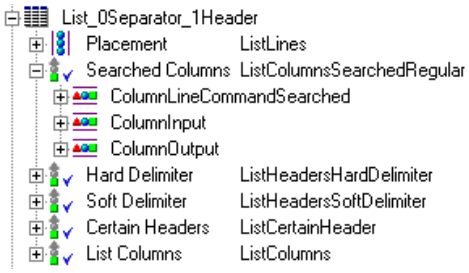


Figure 114. ListColumnsSearchedRegular

The ***ListColumnsSearchedRegular*** Pattern Definition is a OneOf type Pattern Definition. It contains three child patterns:

- ***ColumnLineCommandSearched*** recognizes sequences that begin with an input attribute, continue with 1 or 2 non attribute characters and finish with an output attribute. ***ColumnLineCommandSearched*** is designed to recognize line commands and appears before ***ColumnInput*** so that it takes precedence in the search order. See "Priority Concepts" in *JIS Interface Server: KnowledgeBase User's Guidee* for an explanation of the Pattern Definition search order and its uses.
- ***ColumnInput*** recognizes character sequences beginning with an input attribute and that are input capable.
- *ColumnOutput* recognizes character sequences beginning with an output attribute.

DDS Applications

In DDS applications the location and size of columns in a list are taken directly from DDS information provided by the emulator. Under rare circumstances the *Searched Columns* argument may be used to exclude columns that do not match the Pattern Definition.

By default the *Verify SDF columns using Searched Columns definition* check box in the Pattern Definition's *Parameters* tab is not set.

Figure 115. Verify SDF columns using Searched Columns definition check box

When the check box is set, ACE compares each column received from the emulator with the Pattern Definition attached to the *Searched Columns* argument. Any column that does not match is discarded.

List Headers

ACE contains a special mechanism for recognizing and analyzing list headers, and correctly connecting them to the proper columns. By linking headers to their respective columns ACE is able to recognize:

- More than one header line.
- Headers extending beyond the horizontal borders of the list.
- Headers which are wider than their respective columns.
- Headers belonging to more than one column

A List with Parameters type Pattern Definition contains three arguments, which ACE uses to analyze headers:

1 Certain Headers

2 Hard Delimiter

3 Soft Delimiter

How the Analysis Works

Suppose our host screen contains a list of rock bands and their five best selling albums. The rock bands comprise the headers and the albums are listed in columns C1– C5.

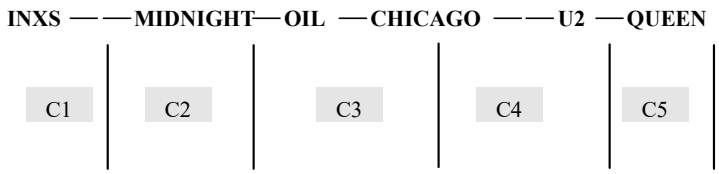


Figure 116. How the analysis works

This list has only one row of headers, where dashes represent blanks that separate text in the header row. The headers are:

- INXS
- MIDNIGHT OIL
- CHICAGO
- U2
- QUEEN

ACE looks along the header row from left to right and first identifies strings that have been defined as *Certain Headers*.

Certain Headers

The *Certain Headers* argument is used to absolutely identify particular character sequences (usually strings) as header elements. Sequences defined as *Certain Headers* are identified as header elements in each list where they occur.

In our example, suppose the string `MIDNIGHT OIL` is defined as a *Certain Header*. In this way it will be found as a header element and eventually linked with one or more columns.

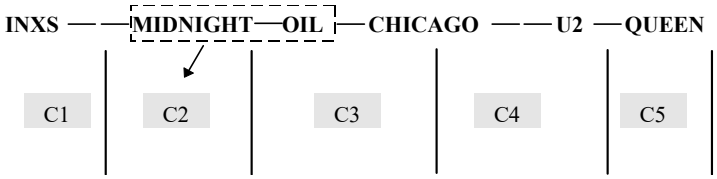


Figure 117. The Certain Headers argument

Hard Delimiter

Next ACE identifies any *Hard Delimiters* and blocks them out. Hard Delimiters, are sequences that are definitely considered non header elements. All remaining elements are defined as header elements.

In the diagram below you can see the sequences defined as header elements so far:

- INXS
- MIDNIGHT OIL
- CHICAGO
- U2 QUEEN

Note that `CHICAGO` is separated from `MIDNIGHT-OIL` on its left because `MIDNIGHT-OIL` is defined as a *Certain Header*, and not through a *Hard Delimiters*.

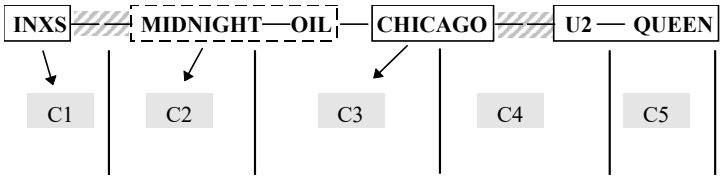


Figure 118. Hard delimiters

Soft Delimiter

To further identify header elements and link them with their appropriate columns, ACE divides header elements isolated by the *Hard Delimiter* into smaller units by using the *Soft Delimiter*. The *Soft Delimiter* identifies and marks non header elements located within header elements that were already isolated by means of the *Hard Delimiter*.

Below you can see that a blank has been identified within the header `U2-QUEEN` by the *Soft Delimiter*. `U2` and `QUEEN` have each been found as header elements.

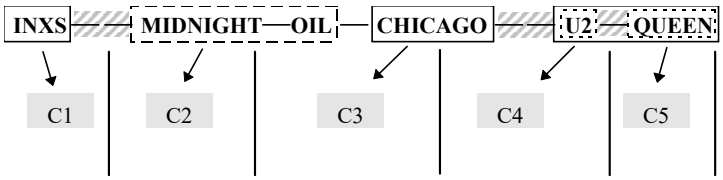


Figure 119. Soft delimiters

Using Certain Headers to Join Header Elements

Had we not defined `MIDNIGHT OIL` as a *Certain Header* from the start, then `MIDNIGHT-OIL-CHICAGO` would have been identified as a Header element by the Hard Delimiter, and subsequently each word as a Header element by the Soft Delimiter, as indicated below.

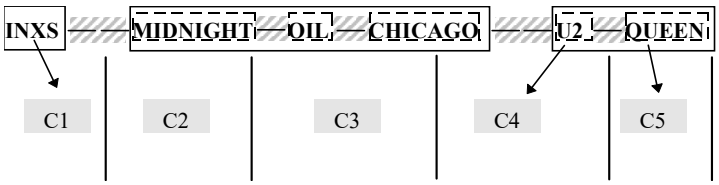


Figure 120. Using certain headers to join header elements

ACE might then have joined the element `OIL` with `MIDNIGHT`, or with `CHICAGO`, or left `OIL` separate and joined `U2` and `QUEEN`. To enforce a specific combination, use the *Certain Headers* parameter to join header elements separated by the Soft Delimiter as above. By defining `MIDNIGHT-OIL` as a Certain Header, ACE will correctly analyze the list.

Linking Headers to Columns

ACE ensures that every header element is linked to a column. It uses the left-right position of header elements relative to columns to determine the correct column(s) to link with each header. ACE also uses the *Soft Delimiters* to split and join header elements according to how well it can subsequently assign columns to header elements.

Hard Delimiter Argument

ACE uses the *Hard Delimiter* to make a first division of the header row into header elements. Characters on different sides of a *Hard Delimiter* are never joined together into a single header element.

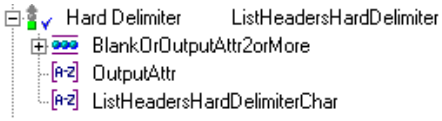


Figure 121. Hard delimiter argument

Example 33. Hard delimiters

Figure 122. Hard delimiter example

In Figure 122, there is one *Hard Delimiter*, the two blanks between `Name` and `Address`. ACE makes a first division of the header row into `Name` and `Address Phone Number`.

Soft Delimiter Argument

ACE uses the *Soft Delimiter* to make a second division of the header row into header elements. Characters on different sides of a Soft Delimiter may or may not be rejoined into a single header element as the analysis proceeds.

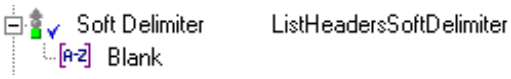


Figure 123. Soft delimiter argument

In the example above there is a *Soft Delimiter* between `Address` and `Phone` and between `Phone` and `Number`. ACE makes a second division of the header row into `Name`, `Address`, `Phone` and `Number`.

Linking the Headers

The last step in ACE analysis is the linking of each header element to a list column according to the relative location of the header elements and the columns. At this stage ACE may decide that a better result is obtained by rejoining header elements separated by a *Soft Delimiter*. In the example above, ACE may decide that `Phone` and `Number` should be rejoined.

Certain Headers Parameter

Certain Headers are header elements that take priority over the division resulting from *Hard Delimiters* and *Soft Delimiters*. Any character sequence that satisfies the Pattern Definition defined as a *Certain Header* will always be analyzed as a header regardless of internal *Hard Delimiters* or internal *Soft Delimiters*, or the position of header elements surrounding them.

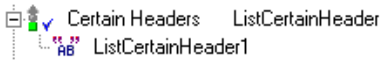


Figure 124. Certain Headers parameter

The string Pattern Definition ***ListCertainHeader*** supplied with ACE is a dummy Pattern Definition that you can modify or replace as necessary. In the example above you might want to ensure that ACE does not consider `Phone` and `Number` as separate header elements. You can do this by adding the string `Phone_Number` to ***ListCertainHeader***.

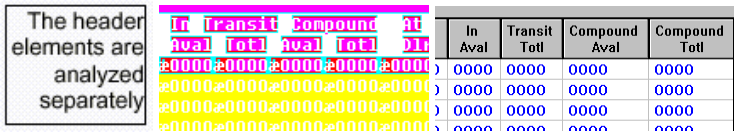
Example 34. Certain Headers

Typical use of *Dertain Headers* is illustrated in the following example.

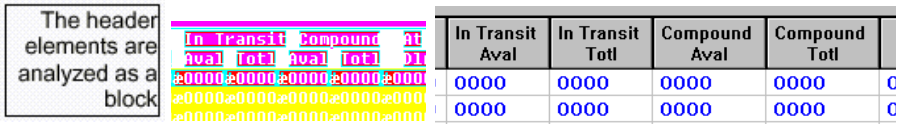
The following figure shows a portion of a list in Analysis and Design Views. The list has a multi-line header. The headers in the top line apply to each of the items in the lower line:

- In Transit Aval
- In Transit Totl
- Compound Aval
- Compound Totl

The header element `Compound` was analyzed as belonging to both of the columns below it, `Aval` and `Totl`. The sequence, `In Transit` should have been analyzed similarly. Instead the ***ListHeadersSoftDelimiter*** Pattern Definition has recognized the blank between `In` and `Transit`, and they have not been rejoined in the further analysis:



When the string `In_Transit` is defined as a *Certain Header*, ACE recognizes the header elements as a block, analyzes them appropriately and provides the desired results:



List Columns

The *List Columns* argument is typically the ***ListColumns*** Pattern Definition. Thus, once ACE has determined that a screen area is actually a column from a list, ACE uses the child patterns of ***ListColumns*** to decide if the area is a column of line commands, or a column of input fields, etc.

Pattern Definitions of List Column Type

A list column is generally composed of a number of similar fields—forming the body of the list column—and a header above. Correspondingly, the ***ListColumn*** Pattern Definitions have two child patterns, one for a column argument and one for a header argument. The Pattern Definitions for these two arguments do not have any inherent list aspects.

Example 35. ListColumnComboBoxViaINI

The Pattern Definition ***ListColumnComboBoxViaINI*** is composed of a Pattern Definition for the header, ***ColumnHeaderTable***, and a Pattern Definition for the column, ***ColumnComboBoxViaINI***.

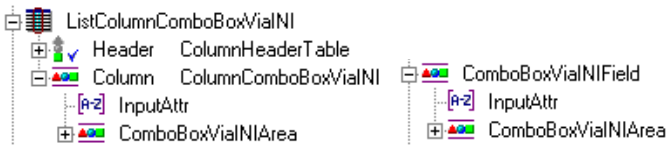


Figure 125. ListColumnComboBoxViaINI

The Column Argument's Pattern Definition

The Pattern Definitions ***ColumnComboBoxViaINI*** and ***ComboBoxViaINIField*** have identical structure and recognize exactly the same character sequences.

The Header Argument's Pattern Definition

A column's header is generally made up of text identifying the purpose of the column. Similarly, a Pattern Definition for the header argument, say ***ColumnHeaderTable*** must be able to recognize general text, composed of any character:

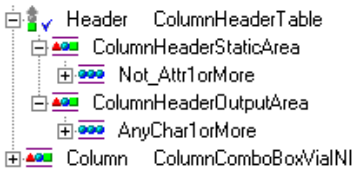


Figure 126. ColumnHeaderTable

The ***Not_Attr1orMore*** child pattern is part of ***ColumnHeaderTable*** for reasons related to how a List with Parameters identifies characters as column headers. ACE uses the List with Parameters arguments *Hard Delimiter*, *Soft Delimiter* and *Certain Headers* to identify column headers. These are the subjects of the next section.

Arranging List Columns and Headers Manually

There are certain instances when you may want to adjust the headers and columns of a list manually. ACE enables you to declare, delete, and resize list headers and list columns. Also, the links between the headers and columns may be viewed, removed and assigned. All of these options are executed using context sensitive menus prompted by the right mouse button. These operations are performed in Analysis View.

When you adjust the links between headers and columns, be aware of the following restrictions:

- All headers must be linked to at least one column. If a header is not linked to a column, and you try to re-analyze a screen, ACE displays the following warning:

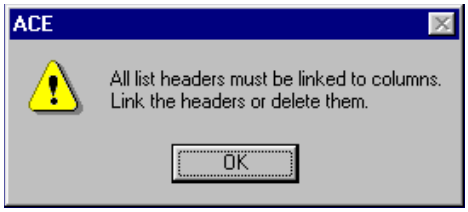


Figure 127. List headers warning

- A header cannot be linked to more than ten columns.

Viewing the Links Between the Headers and the Columns

You can view the links that exist between list headers and their respective columns by right clicking and choosing *View Links*.

To enter View Links Mode:

1. Position the cursor anywhere over the blue portion of the analyzed list.
2. Right click and select *View Links* from the shortcut menu. The analyzed screen displays any links between Headers and their associated Columns. The links appear as yellow lines.

Note: *View Links* can be toggled. To hide the links simply repeat the steps outlined above.

Links Between the Headers and the Columns

Links between headers and columns are presented as yellow lines with square handles. A link extends from the center of a header to the center of its associated column.

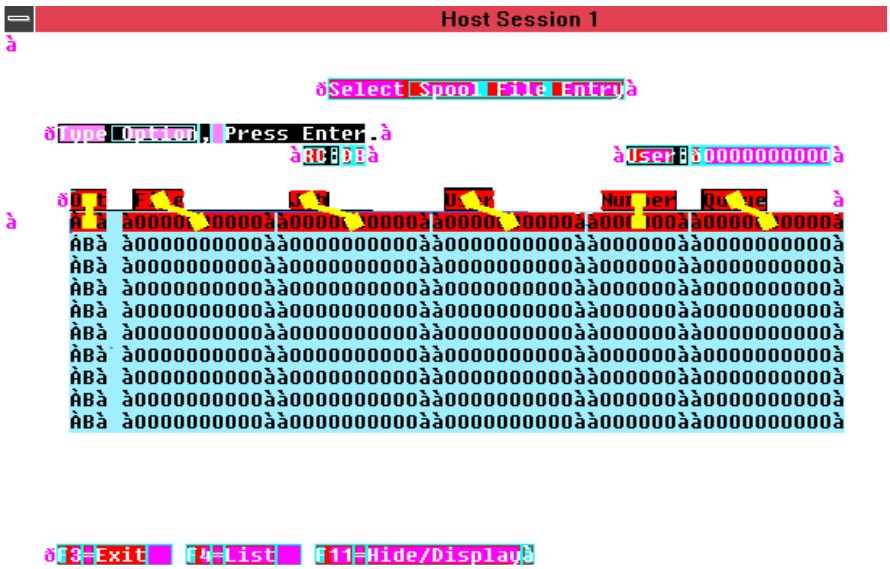


Figure 128. Links between headers and columns

Linking Headers and Columns

On some occasions the arrangement of list headers in relation to each other and their respective columns is not analyzed satisfactorily and requires some manual adjustment. The list headers in the following example demonstrate this:

- On Order Aval
- On Order Totl
- Built Aval
- Built Totl

The relation between the header `On Order` and the header `Totl` has not been found, resulting in the following representation

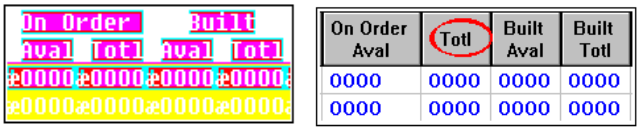


Figure 129. Result of a relationship between header and column not being found

A link is missing between the header `On Order` and the column already linked to `Totl`.

Manually inserting a link between `On Order` and the column solves the problem.

To link a header to a column:

1 Place the cursor on the designated header and click the right mouse button.
2 Select *Link with Column* from the shortcut menu. The cursor changes into a cross-hair.
3 Place the cross-hairs on the first row, colored red, of the designated column and click the left mouse button. a yellow link with handles connects the header and column.

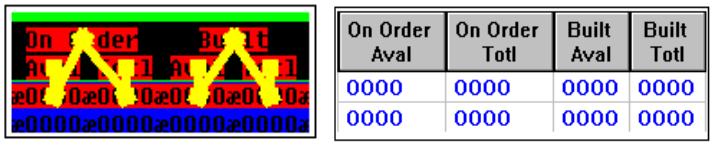


Figure 130. Result of a relationship between header and column being found

4 Click the *Accept* button and re-analyze.

Removing Links between Headers and Columns

To remove links between headers and columns:

1 Place the cursor on or near one of the link's handles.
2 Click the right mouse button and select *Unlink*. The link between the header and column is removed.

Deleting a List Column or a List Header

To delete a list column or a list header:

1 In View Links mode, place the cursor in the red colored area of either the header or the column.
2 Click the right mouse button and select **Delete** from the shortcut menu. The list column or list header is deleted. The selected field is colored blue and its data is disregarded during analysis.
3 From the *Alalysis* menu, select *Apply Analysis Changes*. You cannot undo this operation with the *Undo* button.

Declaring an Area as a List Header or a List Column

Sometimes you need to create entirely new headers or columns, or to redefine the size of current headers and columns. ACE enables you to mark areas and declare them as list headers or list columns.

The area you designate as a new list column must still satisfy the Pattern Definition attached to the *List Columns* List with Parameters argument. In most cases this will already be true, but occasionally you may have to modify the ***ListColumns*** Pattern Definition. One strategy is to add a catch-all Pattern Definition as the last child pattern of ***Listcolumns***.

> **Note:** When redefining an existing header or column, first delete the initial header or column. Then, declare the new header or column

To declare an area as a list header or list column:

1. Press the left mouse button and drag a rectangle around the designated area.
   - For a header, within the header area.
   - For a column, within the first row of the list.
2. Place the cursor within the rectangle and click the right mouse button.
3. Select the *Add list header* or *Add list column* option as appropriate. The designated area is colored red.
4. When you create a new header it must be linked to a column.
5. From the *Analysis* menu, select *Apply Analysis Changes*.

Creating New Columns to the Right

You can mark as a column everything from the right of the cursor up to the next existing column by right clicking and selecting *Add column to the right* from the shortcut menu.

Modifying the Width of a Header or a Column

In cases of inaccurate analysis of a list's columns you may have to modify the location and dimensions of the columns that were recognized. In Analysis View, you can increase or decrease the width of a recognized column, using the values under *Location* and *Dimensions* in the *Pattern Definition Properties* dialog box:

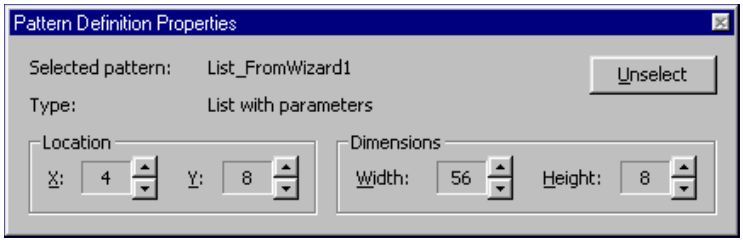


Figure 131. Modifying the width of a header or a column

Location

Location displays the row and column number of the top left hand corner of the currently-selected Pattern Definition.

In the above example, the top left hand corner of the currently selected Pattern Definition is located at column number 4 and at row number 8.

To increase or decrease the width of a column, by moving its left hand border, increase or decrease the *Location* column value.

Dimensions

Dimensions displays the dimensions in rows and in columns of the currently-selected Pattern Definition.

In the above example, the currently-selected Pattern Definition is 56 columns wide and 8 rows high.

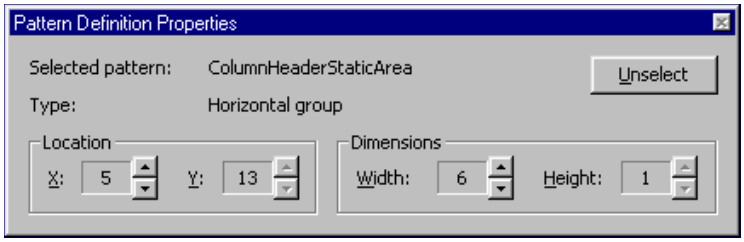
To increase or decrease the width of a column, by moving its right hand border, increase or decrease the *Dimensions* column value.

Splitting a List Header

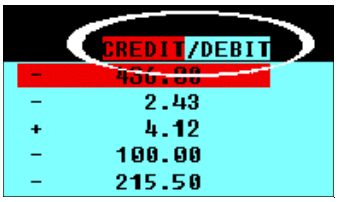
You can split a single list header into two by decreasing the size of the original header area. A second list header is then declared in the area that remains.

To split a list header:

1. In View Links mode, click the designated header to select it. Decrease or increase the size of the header.



2. Decrease the size of the header using the arrows in the *Dimensions* section of the *Pattern Definition Properties* dialog box. The red-colored left side of the original header shrinks, but remains a header, complete with its original links. The blue colored right side of the original header grows.



3 When you have made the desired split, make the blue area of the original header into a new header, using the procedure for declaring an area as a list header.

4 Link all headers to columns.

5 From the *Analysis* menu, select *Apply Analysis Changes*.

Splitting a List Column

Sometimes ACE does not have enough information to distinguish between two columns of data. This results in a single list column containing information from both sources. You can represent such information correctly by splitting the list column in two.

In the following example first and last name details are included in one column:

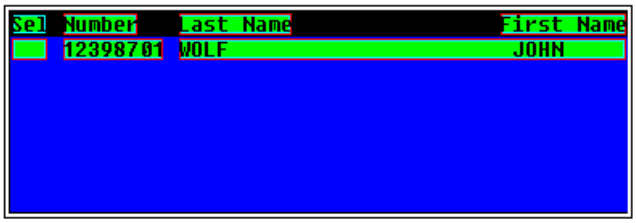
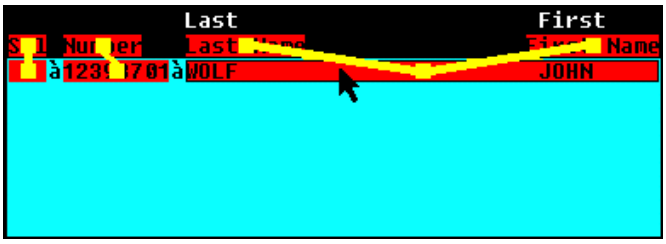


Figure 132. First and last name details are included in one column

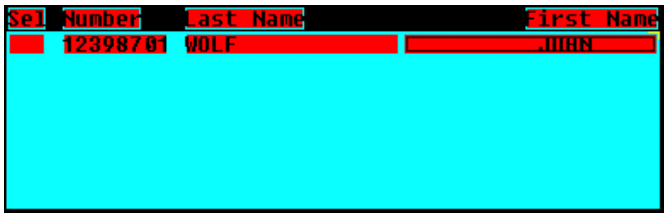
In this case you can split the column and create a separate column for each part of the name.

To split a list column:

1 In View Links mode, position the cursor in the first line of the list column, and right click at the point where the column should be split.



2 Select *Split* from the shortcut menu. The list column is split in two.



3 Update header-column links as appropriate.

4 From the *Analysis* menu, select *Apply Analysis Changes*.

Example 36. Problems in List analysis that can be fixed using the List features

After analyzing a list, the list is presented as follows:

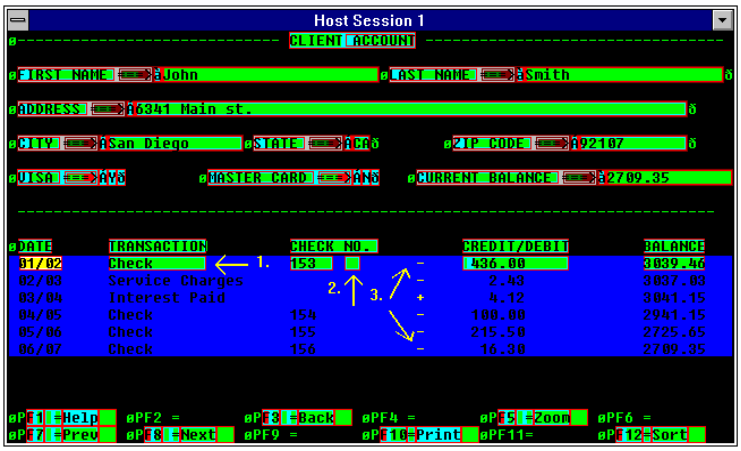


Figure 133. Problems in List analysis that can be fixed using the List features

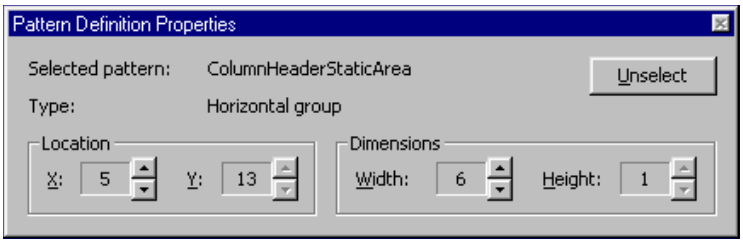
The list in this example has three problematic areas that can be fixed using the list features. The arrows indicate the problematic areas. The first field is a list column, which should be widened to fully include the entries in the column. The second field is a list column that should be deleted, and the third field is a column, which should be included in the list column to its right. ACE allows you to remedy these issues by using the manipulation features for list headers and list columns.

The first and third problems can be fixed by extending the appropriate list column to fully include their respective areas.

Extending a List Column

A list column may be extended to include all the entries in the columns as follows:

1. In *View Links* mode, click the designated list column to select it.
2. Set the *Lock Selection* checkbox of the *Pattern Definition Properties* dialog box.



3. Increase the size of the column using the arrows in the *Dimensions* section of the *Pattern Definition Properties* dialog box.
4. From the *Analysis* menu, select *Apply Analysis Changes*.

Deleting a Column

The second problem can be fixed by deleting the extra column:

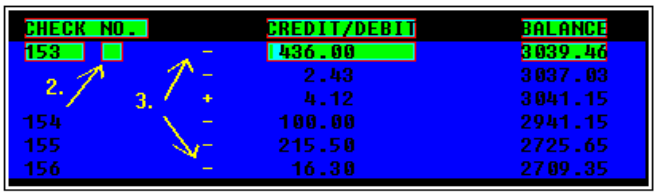


Figure 134. Deleting an extra column

1. In *View Links* mode, right click on the column.
2. From the shortcut menu select *Delete*. The selected area becomes blue.
3. From the *Analysis* menu, select *Apply Analysis Changes*.

Modifying the Type of a Column

In some lists, the particular ***ListColumns*** child pattern that recognized the column's content might not be correct. In such a case, you need to make ACE ignore the incorrect child Pattern Definition and then analyze the screen once again. Upon re-analysis, ACE tries to recognize the list column's content with the ***ListColumns*** child patterns following the ignored child pattern.

The overall effect is to delete the ignored child pattern from ***ListColumns*** for this one column.

To ignore a ***ListColumns*** child pattern:

1. In Analysis View, within the appropriate column of the analyzed list, select the Pattern Definition to be ignored. Use the *F5* and *F6* keys, if necessary, to move up and down the hierarchy of Pattern Definitions.
2. Ignore the Pattern Definition by using the *Ignore* button.
3. Re-analyze. ACE analyzes the one list column as if the ignored child pattern had been deleted from ***ListColumns***.

Types of Lists

Though lists are identified through certain basic definitions, they are not identical in their presentation. ACE can identify three types of lists:

- Standard lists
- Folded lists
- Repeated Columns lists

In addition, you can create your own list layout using existing or new Pattern Definitions. These lists are added to the KnowledgeBase. Using a customized layout is particularly helpful for cases where a number of lists in your application share the same characteristics.

In the following section you will find information about how to work with the existing list types and Pattern Definition related to lists. For a full discussion about how to create new sections and layouts please refer to "Using Layouts" in *JIS Interface Server: Basic User's Guide*.

Standard Lists

A Standard list is the generic term given to a list that can be analyzed without specifying structural parameters aside from the distance between the list records and the header.

To modify the parameters of the Pattern Definition that recognizes the list, open the *KnowledgeBase Definitions* window by double clicking inside the blue-colored list area.

Note: It is strongly recommended that you do not modify one of the existing list Pattern Definitions. Instead you should create a new list type Pattern Definition and re-analyze the list with this new Pattern Definition.

Folded Lists

A Folded list describes a style of presenting a record within a list. Generally, a record is presented in a single row. However, there are some lists that use more than one row to present a single record. These lists are called Folded lists. Some Folded lists can be displayed on the GUI in two ways. Either the list can remain folded, using more than one line for a single record, or the list may be unfolded, presenting all of the information on a single line. Some lists are always displayed folded on the GUI.

Example 37. Folded lists

The following list displays five records and a header row. Each list record, as well as the header row, is composed of two physical lines. The second physical line of each record contains two columns: Schedule and Location. The second line of the header row is immediately below the first line of the header row.

Order	Customer	Request	Warehs	Order Qty	Ship Qty	Alloc Qty	Cur
		Schedule	Location				
2242	1000	7/11/93	SR	67.000			USD
		7/11/93	LOC1				
2243	1000	8/12/93	RP	10000.000			USD
		8/11/93	LOC5				
2365	990099	3/03/94	SR	1.000	1.000		USD
		3/03/94	LOC7				
2369	1	9/03/94	CH	3.000			USD
		9/03/94	LOC9				
2371	1	9/03/94	CH	6.000	6.000		USD
		9/03/94	ASPLOC				

Figure 135. Folded lists in host view

KnowledgeBase Modifications

When your host screen contains a folded list, you should have already chosen an appropriate Pattern Definition, by choosing the corresponding layout, in Layout View.

If there is no appropriate layout/Pattern Definition, because, for example, the number of header and separator rows in your folded list is not covered by the existing Pattern Definitions, create a new Pattern Definition, a new layout, and re-analyze.

If there is an appropriate Pattern Definition, but it is not quite right because, say, the number of lines per record is not correct, then you have two possible approaches:

- If your application contains many folded lists with this different number of lines per record, create a new Pattern Definition with the appropriate parameters.
- If all your lists with this header/separator structure have a different number of lines per record, you can modify the existing Pattern Definition.
  This approach should be used only if you are sure of its overall effect.

Modifying a Folded List's Parameters

You modify the parameters of a folded list Pattern Definition in the *KnowledgeBase Definitions* window's *Parameters* tab:

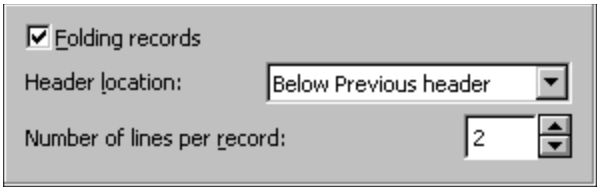


Figure 136. Modifying a folded list's parameters in the Parameters tab

Table 38. Parameters tab options

Option	Description
Folding Records	Specify that a list has folding records by setting this check box.
Header Location	Specify the location of the column headers of the folded lines.
Number of Lines per Record	Specify the number of physical lines per record.

The *Header Location* combo box presents all the possible header locations:

Table 39. Header Location combo box options (Sheet 1 of 2)

Option	Description
Above Previous Header	The column headers are displayed above the first line's headers, each line of headers above the previous line.

Table 39. Header Location combo box options (Sheet 2 of 2)

Option	Description
Below Previous Header	The column headers are displayed below the first line's headers, each line of headers below the previous line (and above the list).
Left of Column Item	The column headers, starting from the second line, are displayed to the left of the column item.
None	The folding lines have no headers. Only the first line has headers.

After specifying the above details, mark the columns of the folding lines in the analyzed screen. When using DDS, the folded columns are identified by ACE automatically. Without DDS you need to assist ACE in identifying the folded columns.

Repeated Columns Lists

In a Repeated Columns list the logical list columns occupy more than one physical screen column. For example, a list whose logical records are made up of three fields normally occupies three physical screen columns. A Repeated Columns version of the same list might occupy six physical screen columns. If the fields are Name, Age and ID#, then each row of the list would contain a Name field, an Age field and an ID# field, corresponding to one logical list record, followed by a Name field, an Age field and an ID# field, corresponding to a second logical list record. The result is that each physical row of the list displays two logical list records.

The header row might also display each column header twice, so that the header row is:

```
Name Age ID# Name Age ID#
```

To properly analyze a Repeated Columns list, a Pattern Definition and layout section must be defined.

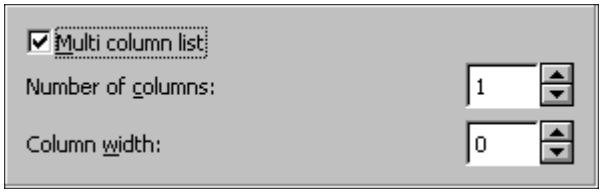
Creating a Pattern Definition for a Repeated Columns List

For ACE to analyze a Repeated Columns list properly, you must create a Pattern Definition for it and a section for the new Pattern Definition.

To create a new Pattern Definition for a Repeated Columns list:

1. Open the *KnowledgeBase Definitions* window in Pattern Definitions View.
2. Create the structure, including child patterns, of a new List with Parameters type Pattern Definition.

   If there is an existing List with Parameters type Pattern Definition with the correct header/separator structure and/or the correct child patterns in the six arguments, you can shorten this process by duplicating the existing Pattern Definition.
3. In the *Parameters* tab set the various options as appropriate. These options have been described previously, except for those dealing with Repeated Columns lists:



4. Set the *Multi column list* checkbox.
5. Set the *Number of columns* field to the number of logical list records per physical screen row.
6. Set the *Column width* to the number of characters between the start of the first logical record in a physical screen row and the start of the second logical record in a physical screen row.
7. In Representation Definitions View create a Representation Definition for your new Pattern Definition. The visible control is a Table that will display the list with one logical list record per table row.

To create a new primary section and layout for a Repeated Columns list:

1. In Layout View create a new primary section, preferably with the same name as your new Pattern Definition.
2. Create a new layout that includes the new primary section.

The easiest way to create this new list layout is to:

1. Apply one of the existing list layouts to a screen.
2. Remove the list section from this layout.
3. Add your new Repeated Columns section to this layout.
4. Save the resulting layout under a new name—again, preferably the same name as the new Pattern Definition/primary section.

List With Parameters Type Pattern Definitions

ACE uses the List with Parameters type Pattern Definition, to analyze lists. As with all Pattern Definitions, a List with Parameters has both properties and structure. These are discussed in the next several sections.

List Pattern Definitions

Lists can have several different formats. To accommodate various formats, the ACE KnowledgeBase currently contains the following List With Parameters type primary Pattern Definitions:

Table 40. List With Parameters type pattern definitions

Definition	Description
ListNoSelection	This primary Pattern Definition is used to recognize a list that does not contain a column of input fields.
ListWithoutHeader	This primary Pattern Definition is used to recognize a list that does not have header rows.
List_NSeparator_MHeader	This primary Pattern Definition is used to recognize a list with *M* header rows and *N* separator rows. Currently, the KnowledgeBase is supplied with Pattern Definitions for lists with 1, 2 or 3 header rows and 0 or 1 separator rows.

Folded Lists

Some lists have their logical records extending over more than one physical row on the host screen. These lists are called Folded lists. The header rows of such lists also extend over more than one physical host screen row. There are two ways that the header row of a folded list can be arranged with respect to the list records:

Table 41. Ways the header row of a folded list can be arranged

Definition	Description
List_Folded_NSeparator_2Header_BelowPrevious Header	This primary Pattern Definition is used to recognize a list whose physical header rows are all physically above the first list record. Currently, the KnowledgeBase is supplied with Pattern Definitions for this type of folded list where there are 0 or 1 separator rows.
List_Folded_NSeparator_ MHeader_LeftOfValue	This primary Pattern Definition is used to recognize a list whose physical header rows are physically intertwined with the first list record. Currently, the KnowledgeBase is supplied with Pattern Definitions for this type of folded list where there are 2 or 3 header rows and 0 or 1 separator rows.

Note: The Pattern Definition name indicates the structure of the list that it is associated with.

You can create new Pattern Definitions of List With Parameters type whose structure reflects other combinations of header rows and separator rows. These and other specifications of the Pattern Definition's structure are made in the Pattern Definition's *Parameters* tab in the *KnowledgeBase Definitions* window's *Properties* pane.

The *Parameters* tab for List With Parameters contains a number of sections. At the moment we will restrict the description to the section that deals with the number of header lines and the number of separation lines.

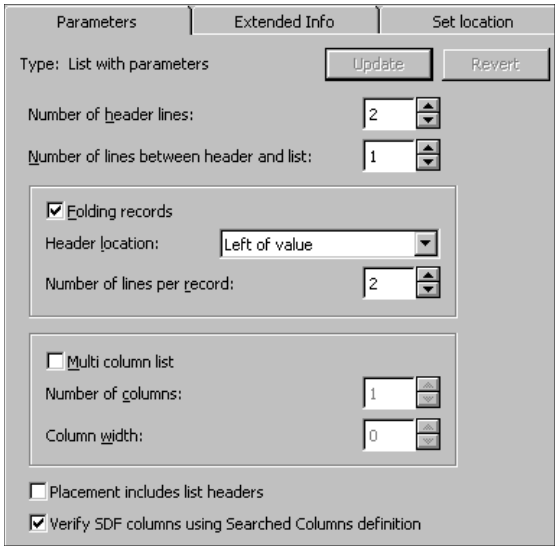


Figure 137. Setting the number of list header and separation lines

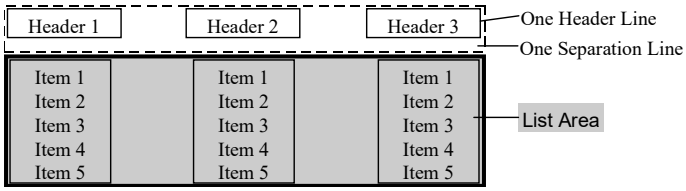
Number of Header Lines

The value listed in the spin box determines the number of lines that are reserved for list headers.

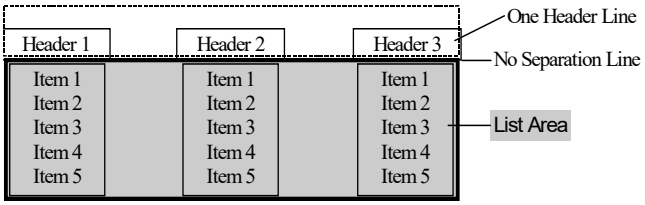
Number of Lines Between Headers and List

The value listed in the spin box determines how many separation lines are placed between the last header line and the first list line. The values are usually 0, 1 or 2. Separation lines are produced by ACE omitting the specified lines from the analysis. While this area may be empty, creating separation lines is a useful way of disregarding information, for instance dashed lines, which appear between the header area and the list.

Example 38. Headers and separation lines



The following screen shows a list with one row of headers and no separation lines between the list headers and the list. Therefore a suitable layout for the list is *List_0Separator1Header*.



Working with Sections

This section describes the procedures for working with sections.

Adjusting List Sections in Panels not From DDS

When adjusting the size of sections within a layout, be certain to size the list section exactly around the actual list. Excess space between the section frame and the actual list may cause the list to be read incorrectly. For more information, see “Placement” on page 220.

The list section is sized exactly around the list.

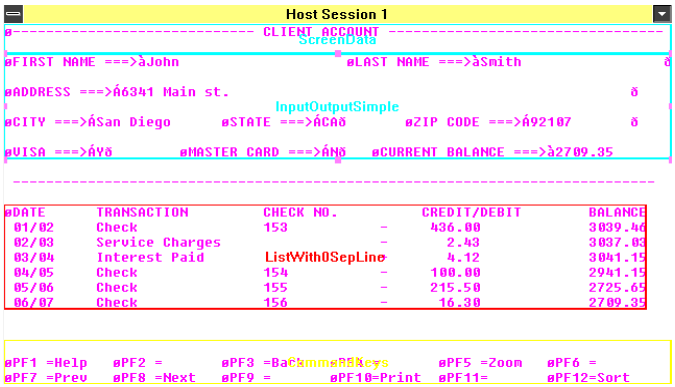


Figure 138. Placement of list sections

Adjusting List Sections in Panels from DDS

When working with DDS support, ACE imposes filters on the screen in addition to screen sections. In this case the list filter within the list section identifies the list area. For this reason the list section need not be adjusted exactly around the list.

Processing Lists Through the Views

This section describes how lists can be processed using the ACE Views.

Lists and Design View

In Design View, every List and List Column type Pattern Definition, and the lower level Pattern Definitions that compose them, must have a Representation Definition.

Note: The **ListColumns** Pattern Definition serves a logical purpose, and does not match an actual pattern on the screen. Therefore, the *ListColumns* Pattern Definition has no Representation Definition.

List With Parameters

The visible component of a List with Parameters type Pattern Definition's Representation Definition is a Table.

The Table control turns host lists into user-friendly spreadsheet-style tables complete with adjustable grid lines and "cut and paste" *Excel* compatibility. The user has access to and can edit every aspect of the spreadsheet including the units, the rows, the columns and the table as a whole, both easily and dynamically.

In runtime, ACE is able to recognize whether a field in a Table column accepts SBCS or DBCS characters, and does not accept input of the wrong kind of characters.

Appearance and Functionality of the Table Control

In runtime, a table appears similar to the following:

	Po	Receipt	Extended Cost	Buyer	Status
1	*	0000002234-MIKE	8000	CST	CLOSED
2	*	0000002235-MIKE	99,000	CST	CLOSED
3	*	0000002236-MIKE	2000	CST	PEND
4		0000002237-MIKE	11,800,000	CST	PEND
5	*	0000002238-MIKE	200	CST	CLOSED
6		0000002239-MIKE	40,000	CST	OPEN
7		0000002240-MIKE	1900	CST	OPEN
8		0000002241-MIKE	35,000	CST	CLOSED

Figure 139. A table as it appears during runtime

The runtime table consists of rows and columns. These rows and columns are made up of cells. Data can be edited on three different levels: by cell, by row or column, or by table. The runtime user edits a unit within the table, by selecting that unit with the mouse. A row or column is edited by selecting the corresponding number or heading. The entire table is selected by clicking on the blank button located in the upper-left corner of the table.

Note: If the user edits one field and moves to a different field, the new data is automatically updated to the host. When the cursor is placed on a static field within the table, the background and foreground colors are inverted.

Selecting Rows and Columns

Several rows or columns may be selected at one time. If the desired rows or columns appear in succession, the user may select them in one of two ways:

- Clicking on the first row or column in the selection and then dragging the cursor to the final selection
- Making an initial selection, holding down the *Shift* key and clicking on the last row or column of the selection.

To select rows or columns which are *not* in succession, for example rows 1 and 6, make a selection, hold down the *Control* key and then click on the other elements of the selection.

The Table Component Style Tab

The following parameters can be set on the *Table Component Style* tab:

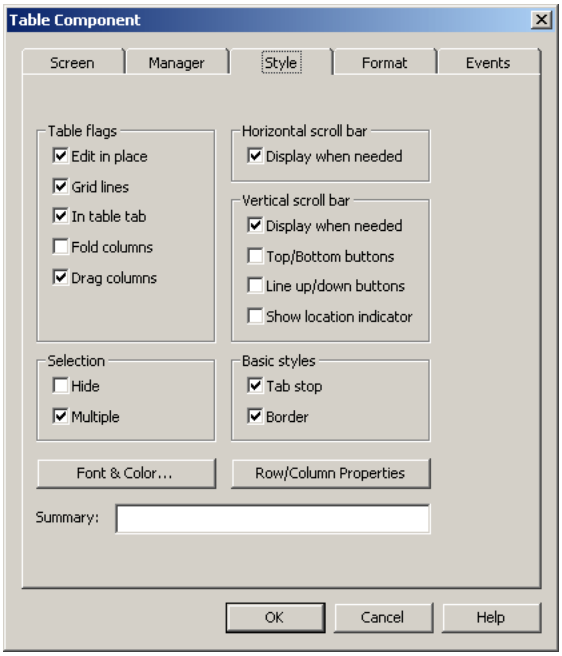


Figure 140. Table component's Style tab

Table Flags

Table 42. Table flags options (Sheet 1 of 2)

Option	Description
Edit in place	Set this check box to edit tables in line. When cleared, the table line is edited in a pop-up window.
Grid lines	Set to display gridlines on the table.
In table tab	When set, the runtime user can tab between the fields of the table.
Fold columns	When a list has been defined as folded, you can set this check box to display the table folded.

Table 42. Table flags options (Sheet 2 of 2)

Option	Description
Drag columns	When set, the user can re-order the columns by dragging them. The feature must be enabled in runtime. In the runtime window, select the **List** tab from the **General Options** dialog box, and set the **Save new column order** check box. **Drag Columns** is disabled when **Fold Columns** is set.

Horizontal Scrollbar

Table 43. Horizontal Scrollbar options

Option	Description
Display when needed	Set to display a horizontal scrollbar when the table is too wide for its display area.

Vertical Scrollbar

Table 44. Vertical Scrollbar options (Sheet 1 of 2)

Option	Description
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 a list to its top or its bottom. When set, buttons are displayed at the top and bottom of the vertical scrollbar. These buttons trigger methods entitled ***GetToBottomOfList*** and ***GetToTopOfList***. These methods are supplied empty. You must add method lines that perform the respective host operations for scrolling a list to its top or its bottom. If the PF8 key scrolls a list to the bottom, then the GetToBottomOfList method should include the line HostType: AidKey: AidPF08 RemainInScreen: False

Table 44. Vertical Scrollbar options (Sheet 2 of 2)

Option	Description
Line up/down buttons	Toggle this option to hide or display the buttons located on the vertical scroll bar which are responsible for advancing the data line by line. By default, the buttons are displayed.
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

Table 45. Selection options

Option	Description
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.

Basic Styles

Table 46. Basic Styles options

Option	Description
Tab stop	When set, allows you to jump to the table using the *Tab* key.
Border	When set, a border is displayed around the table.

Row Properties/Column Properties

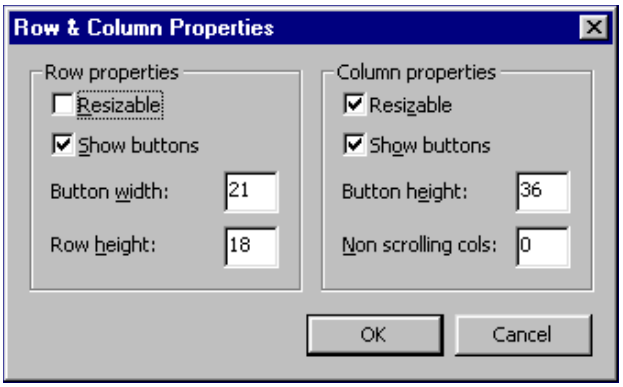


Figure 141. Row and Column Properties dialog box

Table 47. Row and Column Property options

Option	Description
Resizable	When set, the user can resize a row's height, or a column's width, by dragging the table's grid lines. All the rows or columns are resized uniformly.
Show buttons	The *Show buttons* option affects the display of the numerical buttons for the rows, and the column header buttons for the columns. When set, these buttons are displayed in runtime, if cleared they are not displayed in runtime.
Button width	A numerical size you assign to the row buttons.
Row height	A numerical height you assign to the rows.
Button height	A numerical size you assign the column buttons.
Non scrolling cols	You may specify how many columns, beginning with the first, remain stationary (and therefore visible) while scrolling the table horizontally.

Making a Local Modification

When you change a table control's properties locally—by making changes to the *Style* tab from Design View—you may need to click *Apply* before the change takes effect.

For example, by default *Resizable Rows* is not set for any Table control. If you set it for a given table in Design View, you must select *Apply Design Changes*. When you have done this, you will be able to resize rows in Test View, and the end user will be able to resize rows in runtime.

Resizing Columns and Rows

To be able to resize a column or a row in the converter or during runtime, the *Resizable* checkbox in the *Row & Column Properties* dialog box must be set. Changes to column or row size are made in Test View, not in Design View.

To adjust the width of a column, place the cursor in the table's column header area on the line you would like to move. When the cursor is positioned properly, the cursor changes in both color and shape. You may now re-size each column individually.

When you adjust the size of a row, the cursor functions as it does with columns. However, an adjustment made to a single row is uniformly reflected in the size of all remaining rows. Therefore, adjusting the height of a single row automatically adjusts the size of *all* the rows in the table to that height.

Changing Column Order

You can change the order in which a table's columns appear in runtime. In the converter, you perform the following procedure in Test View.

To change the column order in a table:

1. Select the column you wish to move by clicking on its heading button.
2. Hold down the *Shift* key and drag the heading button of the selected column to its new position.

Modifying the Properties of Individual Columns

Tables are made up of a succession of columns, each of which is composed of one type of control. These controls can consist of any of the following:

- Check Boxes
- Combo boxes
- Date controls

- Prompt control
- Adjustable Edit control
- Static control

The Adjustable Edit control and the Static controls are the ones that you are most likely to encounter in your work with tables.

You can modify the properties of a column by modifying the properties of the control that it is made of. The properties of the control are accessible for modification through the Component dialog box of that control, available by double clicking on that control in Design View.

For example, in a table composed of Adjustable Edit or Static controls, you can choose a different font, font color, or background color for different columns in order to better distinguish them during runtime. You can also use one of the *Advanced* options of the Adjustable Edit control such as mask formatting if you wish to apply it to a particular column.

To modify the properties of a column in a table:

1. In Design View, double click on the first cell of the column that you want to modify. The *Component* dialog box of the control is displayed.
2. Choose the modifications you want to apply. Click *OK*.
3. Apply the design changes to see the results.

Note: It is important to double click on the first (top) cell of the column. Clicking on any other cell opens the *Table Component* dialog box instead of the other control's *Component* dialog box.

Lists and Methods

ACE provides several System-Triggered Methods that deal with lists. In runtime these methods enable scrolling through the Table control representing the list.

To modify one of these methods, you open it from the *Design* menu in Design View and add functions in the desired manner.

The methods provided by ACE are:

Table 48. List methods (Sheet 1 of 2)

Method	Description
PageUp	This is a method that scrolls towards the top of a list in the host. The default is the Page Up key.

Table 48. List methods (Sheet 2 of 2)

Method	Description
PageDown	This is a method that scrolls towards the bottom of a list in the host. The default is the Page Down key.
GetToTopOfList	This is a method for scrolling to the top of the list in the host. The method is supplied empty.
GetToBottomOfList	This is a method for scrolling to the bottom of the list in the host. The method is supplied empty.

Note: Currently, each method operates on a whole screen, and not on each list separately. Therefore, if a screen contains more than one list, you cannot operate a different method on each list.

Lists and Runtime Field Information View

When you switch to Runtime Field Information View and place focus on a table the *List Properties* button in the *Decomposition Definition Properties* dialog box becomes enabled.

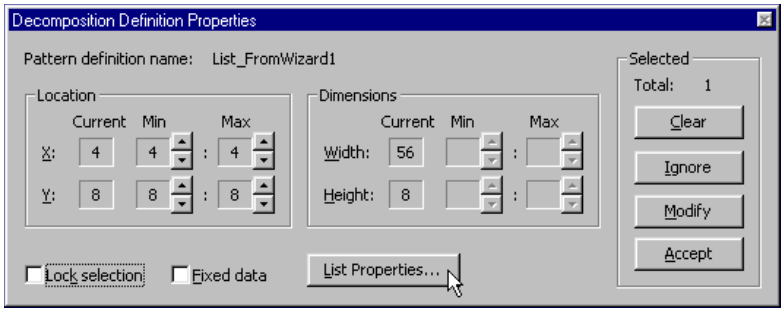


Figure 142. Decomposition Definition Properties dialog box

The *List Properties* button opens the *List Properties* dialog box, in which you can define additional information concerning the way a list should be handled during runtime:

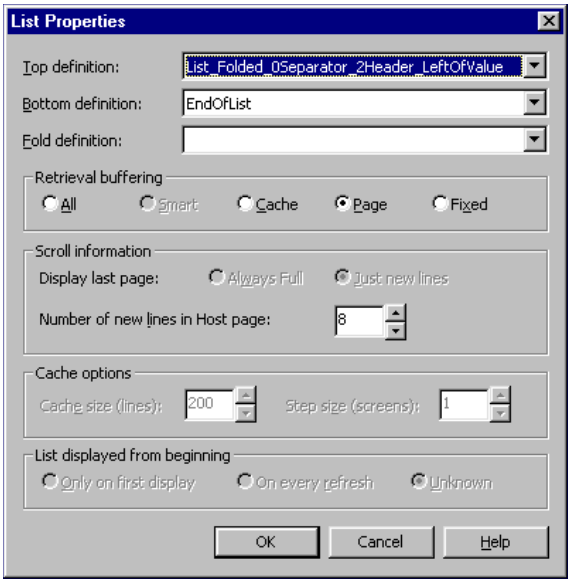


Figure 143. List Properties dialog box

Top and Bottom Definition

In some applications, the top and/or bottom of a list is indicated by a specific pattern.

In Runtime Screen Identification View, you can define specific Pattern Definitions to match such patterns.

The *Top Definition* and *Bottom Definition* combo boxes display the Pattern Definition that matches the top and bottom of the list, respectively.

Bottom Definition has the default Pattern Definition ***EndOfList***.

Fold Definition

When a host list can appear either folded or unfolded, specify a Pattern Definition that ACE uses to differentiate between the two cases. ACE searches the area of the list for the Pattern Definition specified as the Fold Definition. If the Fold Definition recognizes any character sequence in the list area, the list is considered to be folded. Otherwise, the list is considered to be not folded.

The intrinsic properties of the Pattern Definition still apply. Thus, if you know exactly where on the screen the list appears, you can use the *Location* property to focus only on a single physical row. For example, the second physical row might have clearly different structure depending on whether or not the list is folded.

The Pattern Definition you specify as the *Fold Definition* should recognize the first few characters in the second physical list row only when the list is folded. Often, the attribute placement is different:

List Unfolded:

```
àxxxxàxxxàxxxà
àxxxxàxxxàxxxà
àxxxxàxxxàxxxà
àxxxxàxxxàxxxà
```

List Folded:

```
àxxxxàxxxàxxxà
  àxxxxàxxxà
àxxxxàxxxàxxxà
  àxxxxàxxxà
```

The Pattern Definition used as the Fold Definition could be a horizontal group ***Blank2***, ***OutputAttr***, ***NotAttr2***, limited to the beginning of the second physical list row. This Pattern Definition would recognize the characters in its location range only when the list is folded.

Retrieval Buffering

ACE runtime can retrieve a list from the host using one of the following mechanisms:

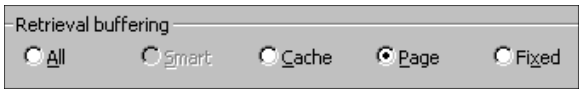


Figure 144. Retrieval Buffering options

Table 49. Retrieval Buffering options

Option	Description
All	The entire list is read before it is displayed to the user.
Page	Retrieves a list page as is from the host.
Fixed	Display fixed sized table as a list.

The most suitable retrieval method depends upon the length of the list and upon the type of the application.

Reading the entire list at once is recommended for relatively short lists for which the retrieval time is a reasonable period of time.

The *Page* mechanism is useful in real-time applications, in which the list values are constantly refreshed. In such applications, the *All* mechanism is inappropriate because the values it displays will be incorrect.

Note: When you have an editable list in which you can only scroll in one direction, use the *Page* mechanism.

Fixed List

You can also use the Expand option to specify a Fixed list.

The Fixed list option is used when you want to display a fixed sized list in table format.

When a screen contains a fixed sized table of input and output fields, it can be represented in a window in two ways. You can either display the table as an array of controls, or you can display it as a fixed table.

The way to display the fixed sized table is a matter of choice.

A fixed table has no vertical scroll bar, unless the list box was made smaller in Design View and not all the records are displayed at once.

List Paging Mechanism

The Page mechanism displays a page of the host list in a table in the window. The Page mechanism is useful in real-time applications, in which the list values are constantly refreshed.

When using the Page mechanism, clicking the arrow buttons in the table's vertical scroll bar scrolls an entire page each time. When the host list is shorter than one page, no vertical scroll bar is displayed.

Figure 145. List paging mechanism options

Note: When using the Page mechanism, the selection of lines in a page is lost when you scroll to a different page in the list

To set Extended List information

1. In Runtime Field Information View, select the list by clicking on it.
2. Set the *Lock selection* check box, in the *Decomposition Definition Properties* dialog box.

3 Click the *List Properties* button.The *List Properties* dialog box opens.
4 Select the *Top definition* that marks the top of the list.
5 Select the *Bottom definition* that marks the bottom of the list.
6 Select the *Fold definition* that marks a folded record.
7 *Retrieval buffering* specifies the way the list is retrieved from the host.
8 Under *Scroll information*, specify the way the last page is displayed in the host: *Always Full* or *Just new lines*.
9 Under *Scroll information*, specify the *Number of new lines in Host page* advanced by each page scroll.

Note: When using the Page mechanism, the number of new lines in host page must be equal to the exact size of the host page, even if there are overlapping lines.

10 Click *OK*.
11 In the *Decomposition Definition Properties* dialog box, click *Accept*.
12 From the Field Information menu bar, choose the RT Decomposition button.

Caching Host List Pages to Create Multi-Page Tables

The Multi Page Table feature increases the runtime performance by reducing the number of times the runtime needs to refer to the host in order to update the table display. This is achieved through the runtime's capacity to load host list pages to the cache and display them in a more economical manner in a table.

The Multi Page Table feature allows you to decide how many host list pages will be cached. You can also determine whether this feature will act globally on the entire Application, or locally on one or more Subapplications.

As a general rule, page caching increases the runtime performance. Note, however, that caching very large numbers of list pages at once may eventually slow down the runtime performance.

Use the Multi Page Table feature to:

- Cache the number of host pages that fit into a single table display.
- Cache multiple sets of host list pages, where each set fits a single table display.
- Change the number of cached host pages according to a multiplier value stated in a DoMethod.

Configuring the *JIS Administrator* can activate the first two options. The third option can be activated through a method that must be written in ACE.

Caching Number of Host Pages to a Single Table Display

You can configure your runtime to cache and display as many host list pages as fit at one time in a single table display.

Example 39. Caching host pages in tables

An enlarged table on a window may have the capacity to display ten rows at a time. When the Multi Page Table feature is enabled and where a host list page contains only five rows, two host list pages will be loaded to cache at once and displayed in the table.

To enable the Multi Page feature:

1. In *JIS Administrator* go to the *Runtime Configuration* tab.
2. From the *Category* combo box, select *List*.
3. Set the *Allow Multi Page Table* check box and click *Apply*.

Caching Multiple Sets of Host List Pages

You can increase the runtime performance by configuring the runtime to cache multiple sets of host list pages at a time. This is achieved by using the *Multi Page Multiplier Value* mechanism in *JIS Administrator*. The number of host pages that are cached is determined by a setting, which acts as a multiplier value.

Example 40. Caching multiple sets of host list pages

If the multiplier is set at two and you need three host pages to fill a single table display, six pages will be loaded from the host and placed in cache to be used in the GUI table. Each time the end-user scrolls up or down out of the cached pages, the runtime refers back to the host and a new set of six pages is loaded to the cache.

> **Note:** This feature is not compatible when lists are displayed as folded in ACE.

To configure a multiplier value on all Subapplication lists:

1. In *JIS Administrator* go to the *Runtime Configuration* tab.
2. From the *Category* combo box, select *List*.
3. Set the *Allow Multi Page Table* check box.

4 In the *Multi Page Multiplier Value* field, type the multiplier number.
5 Click *Apply* to save the configuration.

Note: Unless defined otherwise in ACE - the multiplier value applies to all tables in the Application.

Using a Method to Change the Number of Cached Host Pages

You can write a method in Design View to define the multiplier value.

The global multiplier value set in the runtime environment will apply to all lists unless a call to a ***SetMultiPageMultiplier*** DoMethod is made. When this DoMethod is called, all tables that follow are affected by the multiplier value defined in the DoMethod. The multiplier value can change again if a call is made for another ***SetMultiPageMultiplier*** DoMethod with a different multiplier value configuration.

To configure the multiplier in ACE to change its value during runtime:

1 In Design View, from the *Design* menu select *System-Triggered Method*. The *System-Triggered Methods* manager opens.
2 From the *Methods* list select *General* > ***UserInitSubApplication*** and click *Modify*. The *Define Method* dialog box appears.
3 In the *Line type* list, double click *DoMethod*. The *DoMethod: Method Activation* dialog box appears.
4 From the *Full List* of methods, select the ***SetMultiPageMultiplier*** DoMethod.
5 Click *Assign Values*. The *Method Parameters* dialog box appears.
6 In the *Parameters value* input field type the multiplier factor (must be a digit).
7 Click *OK* to save the configuration.
8 Click *Close* in the System-Triggered Methods dialog box to save the configuration.

Lists and Runtime Screen Identification View

In this view, the whole list (except the headers) has to be marked as Variable, i.e. marked in red.

Chapter 18. External Data

In a running application, the information at the application's disposal is derived almost solely from the host. However, if you wish to supply your application during runtime with yet more information from an external data source, it is possible to do so by opening another session from within the running application.

The JIS Server can open an additional host session to another application or establish a connection with an external database. This greatly increases the span of your application to encompass more sources of information, ultimately granting the end-user access to more information than the host can provide.

This chapter describes:

- External Data Source Connectivity
- DBSession Connectivity
- HostSession Connectivity
- Troubleshooting ExternalData Methods

This chapter also presents you with a list of the relevant DoMethods and their properties.

External Data Source Connectivity

External Data Source Connectivity (EDSC) is a feature that allows you to open an additional session to an external data source, extract and manipulate information from this external data source, and close this additional session.

The additional session may be one of the following:

- *DBSession:* for connecting to an external database.
- *HostSession:* for connecting to an additional host session.

External Data Methods

External Data Source Connectivity is governed by methods known as ExternalData methods.

The following actions are set within an ExternalData method:

- Opening the additional session.

- Retrieving or inserting the desired information.
- Closing the session.

ExternalData DoMethods

The following DoMethods are executed by ExternalData:

Table 50. ExternalData DoMethods (Sheet 1 of 2)

DoMethod	Description
AllocDBSession	Establishes a connection with an external database. *Returns:* Reference to the DBSession interface in case of success, or null in case of failure. *Parameters:* url - Enter a database URL in the following form: jdbc:subprotocol:subname or subprotocol:subname userID - Enter the database username on whose behalf the connection is being made. password - Enter the user's password.
AllocHostSession	Opens an additional host session. *Returns:* Reference to the HostSession interface in case of success, or null in case of failure. *Parameters:* host - Enter the IP address or host name of the JIS Server. port - Enter the RMI registry port used by the JIS Server. application - Enter the name of the Application installed on the JIS Server. applProfile - Enter the name of the application profile installed on the JIS Server.

Table 50. ExternalData DoMethods (Sheet 2 of 2)

DoMethod	Description
ConnectToDBSession	Retrieves a previously allocated DBsession. *Returns:* Reference to the DBSession interface in case of success, or null in case of failure. *Parameters:* SessionHandle - Enter the session handle.
ConnectToHostSession	Retrieves a previously allocated host session. *Returns:* Reference to the HostSession interface in case of success, or null in case of failure. *Parameters:* SessionHandle - Enter the session handle.
GetLastError	Retrieves the value of a database table cell as an integer by the name of the cell's column in the table's current row. *Returns:* A string that represents the last exception that has been thrown by either of the ExternalData, HostSession or DBSession methods. If the last invocation was success, then it returns null. *Parameters:* This DoMethod has no parameters.

DBSession Connectivity

When a DBSession method is triggered, the JIS Server establishes a connection with an external database and sends a predefined SQL query to the database. In this session, the query produces a table containing the information that fits the query specifications. Then, using a series of ***Get*** DoMethods, the data is retrieved from specified rows and columns.

In order to conduct such a query and navigate through the database table to retrieve or insert data, you must use a special set of DoMethods called DBSession DoMethods.

Initial Settings for DB Sessions

In order to work with DB sessions, certain preparations must be made. These being:

- Installing the appropriate driver.
- Modifying the jacadasv.ini file.
- Updating the jacadasv.bat file.

To install the driver:

1. Obtain a driver file. You might want to consult with your database vendor on this topic. It is important that you obtain a driver that is ideal for your type of database. Otherwise, you can install a driver for a JDBC:ODBC bridge.
2. Open the driver into the `<InstallDir>\JacadaFiles\classes` directory. This creates a whole set of directories containing libraries of Java classes.

To modify the jacadasv.ini file:

1. Using any text editor open the jacadasv.ini file located in the `<InstallDir>\JacadaFiles\classes\` directory.
2. In the *.ini file, create a new section named `[ExternalData]`
3. Under the new [ExternalData] section, insert the class path in the following manner:

   ```
   [ExternalData]
   DataBaseDriverList=<FullPathToDriverFiles>
   ```

   The syntax of the path is comprised of the subdirectories under the `<InstallDir>\JacadaFiles\classes\` directory, in such a way that the subdirectory levels are separated from each other with a period "." character.

To update the jacadasv.bat file:

1. Open the jacadasv.bat file located in the `<InstallDir>` directory.
2. In the batch file, insert the following path:

   ```
   <InstallDir>\JacadaFiles\classes
   ```

Example 41. Querying an iSeries DB2 database

To query an iSeries DB2 database, you can install jt400.jar on your computer. The jt400.jar archive file contains the iSeries Toolbox for Java and JTOpen. Follow these steps:

1. Download jt400.jar from the IBM web site at *http://www-1.ibm.com/servers/eserver/iseries/toolbox/downloads.htm*
2. Unzip jt400.jar into the `<InstallDir>\JacadaFiles\classes` directory. This creates `<InstallDir>\JacadaFiles\classes\com\ibm\<LibName>`

3 In the jacadasv.ini file, create the new section `[ExternalData]` as follows:

```
[ExternalData]
DataBaseDriverList=com.ibm.as400.access.AS400JDBCDriver
```

4 In the jacadasv.bat file, insert the following path:

```
<InstallDir>\JacadaFiles\classes
```

DBSession DoMethods

The AllocDBSession and ConnectToDBSession DoMethods enable the use of the DBSession DoMethods. These DoMethods become available in the Method Editor's DoMethod dialog box only when referenced to an AllocDBSession or a ConnectToDBSession method line.

Table 51 shows the DBSession DoMethods:

Table 51. DBSession DoMethods (Sheet 1 of 3)

DoMethod	Description
Close	Closes the current DB session.
ExecuteQuery	Executes an SQL statement in the database. *Returns:* Table containing all of the rows that match the SQL statement specifications, the row count for INSERT, UPDATE or DELETE statements, 0 in case the query returns nothing, and -1 in case of failure. *Parameters:* query - Enter the SQL statement.
GetHandle	Gets the current DB session's handle. *Returns:* The session handle.

Table 51. DBSession DoMethods (Sheet 2 of 3)

DoMethod	Description
GetIntByColumnIndex	Retrieves the value of a database table cell as an integer by the index number of the cell's column in the table's current row. *Returns:* The cell's value in case of success and 0 in case of failure. *Parameters:* columnIndex - Enter the column's index number.
GetIntByColumnName	Retrieves the value of a database table cell as an integer by the name of the cell's column in the table's current row. *Returns:* The cell's value in case of success and 0 incase of failure. *Parameters:* columnName - Enter the column's name.
GetStringByColumnIndex	Retrieves the value of a database table cell as a string by the index number of the cell's column in the table's current row. *Returns:* The cell's value in case of success and null in case of failure. *Parameters:* columnIndex - Enter the column's index number.
GetStringByColumnName	Retrieves the value of a database table cell as a string by the name of the cell's column in the table's current row. *Returns:* The cell's value in case of success and null in case of failure. *Parameters:* columnName - Enter the column's name.

Table 51. DBSession DoMethods (Sheet 3 of 3)

DoMethod	Description
Next	Moves the cursor to the next row in the database's table. *Returns:* _TRUE if the next row is a valid row. _FALSE if there are no more rows in the table.

Creating a DBSession Method

In most cases a DBSession method opens a DB session, queries the database, retrieves information, manipulates the retrieved data and closes the session.

To create a DBSession method:

1. Open a DB session using the AllocDBSession or the ConnectToDBSession DoMethod.
2. Query the database using the ExecuteQuery DoMethod.
3. Move the cursor down one row using the Next DoMethod.
4. Retrieve data from the row using one of the Get DoMethods.
5. Repeat steps 3-4 as needed. This is best performed using a Do loop.
6. Manipulate the retrieved data according to your needs.
7. Close the DB session using the Close DoMethod.

Retrieving Data

After a query has produced a table with the specified data, the cursor is placed on the left-hand side of the table's heading row. The Get DoMethods operate only on the "current row", that is the row in which the cursor is currently placed. Therefore, you must first insert a Next DoMethod to move the cursor down one row, from the heading row to the first row of retrievable data. This must be done even if the query has produced only one row. A Get DoMethod requires the name or index number of a column, and returns the content of the specified column in the current row, as a String or as an Integer.

To "get" entries from ALL of the rows, you must repeat the Get method line(s) for each and every row. This can be done with the use of a Do loop containing a Next method line followed by the Get method line(s).

Manipulating the Retrieved Data

There are several possible uses for the retrieved data. You may choose to display the retrieved data in a Message Box. You can also insert the retrieved data into an input field, or "write" the data into the variable pool and use it as another DoMethod's parameter, or even save it as a shared user variable to be used in a subsequent Subapplication.

Inserting Data into the Database

To insert data into a database, use the ExecuteQuery DoMethod as well. Provide the DoMethod with an SQL statement that writes into the database, such as INSERT, UPDATE or DELETE statements. In such a case, the DoMethod returns the row count.

Retrieving a Previously Allocated DB Session

When you open a new DB session, use the GetHandle DoMethod referenced to the AllocDBSession method line. GetHandle returns the session's handle. Store the handle as a shared user variable. This being done, you can later retrieve the session's handle from the variable pool and use it with the ConnectToDBSession DoMethod to retrieve the previously allocated DB session.

Example 42. Opening a DB session with parameters:

The following method opens a DB session with these parameters:

url: "jdbc:as400://135.12.81.2;libraries=AA,*LIBL"
userID: "sefi"
password: "sm777".

1. The method queries the database, selecting the "MYFILE" table from the database.
2. Then, within a Do loop, a string and an integer are retrieved from two separate columns of each row and displayed in a Message Box.
3. In each instance of the Do loop, the cursor goes down one row, and a new message box appears with new results.
4. The Do loop is broken when the Next DoMethod returns _FALSE, after the cursor reaches the bottom row.
5. After this, the DB session is closed.

```
#0 = DoMethod: Receiver: `ExternalData` Method: AllocDBSession
Parms: ( `"jdbc:as400://135.12.81.2;libraries=AA,*LIBL"` , `"sefi"` ,
`"sm777"`  )
DoMethod: Receiver: `#0` Method: ExecuteQuery
```

```
Parms: ( `"SELECT * from MYFILE"`  )
Do: Times: `50`
   #1 = DoMethod: Receiver: `#0` Method: Next
   Parms: (  )
   If: Cond: ` ( #1 ==  _FALSE  ) `
   Break:
   EndIf:
   #3 = DoMethod: Receiver: `#0` Method: GetStringByName
   Parms: ( `"APP"`  )
   #2 = DoMethod: Receiver: `#0` Method: GetIntByIndex
   Parms: ( `4`  )
   #4 = DoMethod: Receiver: `#2` Method: AsString
   Parms: (  )
   MsgBox: MsgBoxType: Info Message: `#3 + " " + #4`
   Caption: `"Query Result"` Default: 1
EndDo:
DoMethod: Receiver: `#0` Method: Close Parms: (  )
Return: NoValue
```

HostSession Connectivity

When a HostSession method is triggered, the JIS Server opens a second host session. In this additional session, you can call for another application, or even for another separate session of the same application. In this session you can navigate between the different screens of the application, “put” variables into fields or tables, “get” variables from fields and tables, and execute methods. All this is done using a special set of DoMethods called HostSession DoMethods.

HostSession DoMethods

The AllocHostSession and ConnectToDBSession DoMethods enable the use of the HostSession DoMethods. These DoMethods become available in the Method Editor’s DoMethod dialog box only when referenced to an AllocHostSession or a ConnectToHostSession method line.

Table 52 shows the HostSession DoMethods:

Table 52. HostSession DoMethods (Sheet 1 of 2)

DoMethod	Description
ExecuteMethod	Executes a method without parameters on the JIS Server. Upon return the screen data is updated. *Returns:* _TRUE in case of success and _FALSE in case of failure. *Parameters:* methodName - Enter the method name.
Free	Frees the additional host session.
GetCurrentScreenNam e	Gets the name of the current screen on the additional host session. *Returns:* The screen name.
GetHandle	Gets the additional host session's handle. *Returns:* The session handle.
GetTableCell	Retrieves contents of a specified table cell. *Returns:* The value of the table cell or null if the cell is not found *Parameters:* tableName - Enter the name of the screen table. row - Enter the row number of the table cell. colName - Enter the name of the table cell's column.

Table 52. HostSession DoMethods (Sheet 2 of 2)

DoMethod	Description
GetVar	Retrieves contents of a specified host screen field. *Returns:* The value of the screen field or null if the field is not found. *Parameters:* `screenFieldName` - Enter the screen field's name.
PutTableCell	Inserts a variable into a specified table cell. *Returns:* The previous value of the table's cell in case of success or `null` in case of failure. *Parameters:* tableName - Enter the name of the screen table. row - Enter the row number of the table cell. colName - Enter the name of the table cell's column. itemValue - Enter the new value of the table cell.
PutVar	Inserts a variable into a specified host screen field. *Returns:* The previous value of the screen field or null in case of failure. *Parameters:* `screenFieldName` - Enter the screen field's name. screenFieldValue - Enter the screen field's value.

Creating a HostSession Method

In most cases a HostSession method opens an additional host session, navigates through the application's screens, retrieves information, manipulates the retrieved data, and frees the session.

To create a HostSession method:

1. Open an additional host session using the AllocHostSession or the ConnectToHostSession DoMethod.
2. Navigate through the application's screens.
3. Retrieve information using the Get DoMethods.
   -AND/OR-
   Insert information using the Put DoMethods.
4. Manipulate the retrieved data according to your needs.
5. Repeat steps 2-4 as needed.
6. Free the additional host session using the Free DoMethod.

Navigating Through the Application

Normally, during runtime you navigate through an application by pressing the Enter key or the FKeys and by entering command lines, as in a SignOn screen. In this case it is the same, only here you must program the HostSession method to perform these actions.

For pressing a key, insert an ExecuteMethod DoMethod and provide the appropriate method name as the DoMethod's parameter. Use the Enter method for pressing the Enter key or one of the Fkey methods for pressing an Fkey.

For entering a command line, insert a PutVar DoMethod and provide the name of the field and the appropriate command line as the DoMethod's parameters. Following the PutVar method line, insert an ExecuteMethod DoMethod with the Enter method as the DoMethod's parameter.

Inserting and Retrieving Data

After navigating to the desired screen, use the HostSession DoMethods to retrieve and insert the desired information.

Supply the HostSession DoMethods with the appropriate parameters. Provide the screen field names for fields, or table names, column names and row numbers for table cells. The Put DoMethods also require the new values you wish to insert.

Note: The Put DoMethods return the previous value of the field.

Retrieving a Previously Allocated Host Session

When you open a new host session, use the GetHandle DoMethod referenced to the AllocHostSession method line. GetHandle returns the session's handle. Store the handle in a shared user variable. This being done, you can later retrieve the session's handle from the variable pool and use it with the ConnectToHostSession DoMethod to retrieve the previously allocated host session.

Troubleshooting ExternalData Methods

For troubleshooting an ExternalData method, use the GetLastError DoMethod. This DoMethod returns the last error (if any) in the ExternalData method.

To use the GetLastError DoMethod:

1. In the ExternalData method, insert a GetLastError DoMethod method line executed by ExternalData. You should insert the new method line below the method line in which you suspect there is an error.
2. Under the GetLastError method line, insert a MsgBox method line. Set the message box to display the GetLastError DoMethod's returned value.
3. Generate runtime.
4. Run the Application.
5. Trigger the ExternalData method.

If a message box appears displaying the type of error, then this means that there is an error in a method line preceding the GetLastError method line.

If, on the other hand, a message box with content does not appear, then this means that the method line containing the error is after the GetLastError method line. In such a case, you should repeat the process, inserting the GetLastError method line into a position further down in the method. By repeating the above process, moving the GetLastError method line's position further down each time until the message box displays the error in runtime, you should be able to discover the exact location of the error as well as its type.

Part VI. About Using *.ini Files to Increase Efficiency

This section explains how ACE *.ini files are used.

ACE contains two default *.ini files, one converter.ini file and one `<ApplName>.ini` file. The `<ApplName>.ini` file is created the first time that runtime is generated. More *.ini files can be added as continuations of the main *.ini files. Editing the SDF.ini or DDS.ini files can aid screen analysis by forcing strings into fields.

More information about *.ini files can be found in other sections of the ACE documentation. Consult the Index for *.ini files.

Chapter 19. Using *.ini Files

*.ini file settings affect the GUI design and/or runtime behavior of the Application. Reconfiguring *.ini file settings makes changes globally and avoids recompiling the entire Application.

Great caution is advised when editing or adding entries.

This chapter includes:

- Using Multiple *.ini Files
- Using DDS.ini and SDF.ini
- Entries in <ApplName>.ini
- Converter.ini File
- Translating Accelerator Keys

Using Multiple *.ini Files

A *Windows* *.ini file is limited to 64K. Occasionally, an *.ini file may contain many entries and thus exceed this size. As *.ini files grow, the capacity to access them in an expedient manner may decrease. ACE resolves this issue by enabling multiple *.ini files. ACE can then be set to load the multiple *.ini files into the system cache memory. ACE then accesses them as if they were a single file.

Multiple *.ini Files

Multiple *.ini files are continuations of the main *.ini file. ACE uses two main *.ini files—one for the converter (cnvrt400.ini or convert.ini) and one for the runtime (application.ini). Additional files can be named according to the user's discretion, but must always have the *.ini extension. Additional files must always be loaded into the system cache memory to be used by ACE.

When your *.ini file gets too large, create multiple *.ini files by:

1. Copying a portion of the original *.ini file into the new file
2. Saving the new file in the appropriate directory with an .ini extension

   Once the Main (parent) *.ini and the extension (child) *.inis are loaded into the cache, ACE reads the data as if it were contained in a single *.ini file.

Note: Data belonging to a single section can be distributed in various *.ini files. However, the section name must appear in both files. When the data is loaded into the system cache, ACE handles the data in the different files as a single file.

Loading Files into the Cache Memory

Specify the files to be loaded into the cache through the main *.ini file. Type the following three lines into the main *.ini file:

```
[INICache]
UseCache=1
LoadFiles= <full path\File1>;<full path\File2>;etc...
```

- The *first* line opens an *.ini cache section.
- The *second* line activates the cache memory and loads the files indicated in the previous line. To enable this feature change the flag to "1." The default setting for this feature is OFF (UseCache =0).
- The *third* line lists which files will be loaded into the cache memory.

Set the UseCache flag to 1 before starting the runtime application. The multiple *.ini files will be read into the system cache memory when starting an application.

You can also update the cache dynamically, when a runtime application is running.

Updating the Cache Dynamically in Runtime

If you are working in a runtime application, you can update the values in your CnvtrINI file while it is memory resident.

To do so, in ACE, add a GUI control that triggers a method capable of reading the files listed in the `LoadFiles` line of the *.ini file.

Use the following method:

Receiver: System
Method: ***ReloadApplIniFile***

Update your INI files while your runtime application is running. Then, click the control you added to update the values to the system cache memory.

Using DDS.ini and SDF.ini

This section explains the usage of the DDS.ini and SDF.ini files.

Using the DDS.ini

By editing the DDS.ini you can force character strings into a field. This is typically done to aid analysis when a default string is not recognized by any of the Pattern Definitions. There may be times when an application has both DDS files and SDF files. This section refers to the DDS files only. For information concerning SDF files, see "Using the SDF.ini" on page 285.

The DDS.ini is divided into sections. Each section is identified by brackets [ ] and the section name is placed within the brackets. Section names are not case sensitive. Entries in the brackets can be placed in a Subapplication-specific section or in a Global section. If entries are placed in the Global section they affect all the Subapplications in the Application or library. If an entry for a particular field is placed both in the Global section and in the Subapplication specific section, the value entered in the Subapplication specific section takes priority.

> **Note:** You cannot have a section named Global and a Subapplication named Global in the DDS.ini. This **is** possible in the SDF.ini.

As an example, let's say that a string with the format `FIELD=NewString` is located under a Section name. The key, placed to the left of the equals sign, represents the field name and is case sensitive. The value, placed to the right of the equals sign, represents the new string and is case sensitive. The key/value string applies only to the Section name it is placed under.

> **Note:** It is possible for the new string to contain equals signs. For example:
> `Field=F3=Exit F4=Next`

Editing the DDS.ini

The DDS.INI is located in the DDS directory of the Subapplication's Application or library. Call the DDS.INI using a text editor. Ensure that there is at least one blank line below the file's last entry. After you have saved your changes in the INI file close ACE and then re-open ACE. Closing and re-opening the program restarts the converter so that your changes can take effect. From the *File* menu, perform *Maintain Screen Images* on the Subapplication. The new string appears in the Subapplication.

To force a character string into a field configure the DDS.ini as follows:

[Options]	Default DDS.ini text
European Date=0	Default DDS.ini text
Remove Protections=0	Default DDS.ini text
Ignore CLRL=0	Default DDS.ini text
[Subapplication1]	Subapplication specific section name - specifies a Subapplication. The Subapplication's name is found in the Title bar of the ACE window.
FIELD1=String1	FIELD1 - represents the name of the field in the Subapplication. To find the field name double click on the field in Design View. Click the *Manager* tab. The field name is displayed in the *Variable name* field. String1 **-** represents the new string and is case sensitive. **Note:** The field name has a DDS prefix. Disregard the DDS prefix when entering the name in the DDS.INI. This entry is case sensitive.

Note: ACE automatically corrects illegal DDS names and strings. Thus, the name specified in the *Manager* tab may not be the full name of the DDS field. For example, a DDS field with the name, `#C@A78` is corrected by ACE to `CA78`. When referencing DDS field names in the *.ini files, the original DDS field names, as they appear in the iSeries, must be used

[Global]	Entries in the panel's Global section are applied to all the panels in the Application or library. Entries in the Subapplication Specific sections override the settings in the Global sections.
FIELD1=String1	

Using the SDF.ini

By editing the SDF.ini you can force character strings into a field. The SDF.ini can identify a field by its name or by its location. This is typically done to aid analysis when a default string is not recognized by any of the Pattern Definitions.

The SDF.ini is divided into sections. All valid section names have the format, `[Subapplication By Name]` or `[Subapplication By Place]`

Forcing a Field's Contents by Name

To force a field's content by the field name you add a line with the format of `name=value` to the appropriate `By Name` section in the *.ini file. `Name` is the name of a specific Subapplication or the string `/Global/`. The slashes are mandatory and ensure that you can use the name `global` for a panel. Names of fields are case sensitive. Names of sections either by name or by place are not case sensitive.

Forcing a Field's Contents by Place

To force a field's contents by its location you add a line with the format of `place=value` to the appropriate `By Place` section in the *.ini file. The place of a field is represented by a string of the format, `row/col/length`. If the height is more than 1 the string should be `row/col/length/height`. The row and col values are the 1-based coordinates of the field's attribute. Length is the field's length and does not include the attribute. If any of the three do not match the current field's location, its contents are not replaced. Long initializers are truncated to fit into the field.

To place the cursor in a given location, you can use either the `By Place` or `By Name` section. If both are specified, `By Name` takes precedence. This holds true for all entries. Add lines such as `Cursor Row=row` and `Cursor Col=col` to the appropriate section, where `row` and `col` are the 1-based coordinates of the desired cursor location.

Editing the SDF.ini

The SDF.INI is not automatically created by ACE. Using a text editor you must write the SDF.INI and save it in the Application's SDF directory. Ensure that there is at least one blank line below the file's last entry. After you have saved your changes in the INI file close ACE and then re-open ACE. Closing and re-opening the program restarts the converter so that your changes can take effect. Perform *Maintain Screen Images* from the *File* menu in the Subapplication. The new string appears in the Subapplication.

To force a character string into a field configure the SDF.ini as follows:

```
[/Global/ By Name]
USERID=E4321


[/Global/ By Place]
1/1/8=SCREENID

[MAP001 By Place]
1/72/8=1-Dec-1998
   //This places 1-Dec-1998 in the field located at 1/72/8
Cursor Row=15
Cursor Col=55

[MAP001 By Name]
USERID=BILL
   //This overrides the Global setting above.
FKEYS=F3=Exit F5=List F7=Up F8=Down
```

Entries in <ApplName>.ini

This section covers the entries that can be made in the `<ApplName>.ini` file. The `<ApplName>.ini` file is the Runtime *.ini file.

<ApplName>.ini

The first time you generate a runtime, ACE creates an *.ini file that contains settings specific to the Application. This *.ini file has the same name as the Application, and is saved to the Application's rt32 directory. This section refers to this file as `<ApplName>.ini`. This same file is also sometimes referred to as the Application *.ini file or the runtime *.ini file.

Example 43. <ApplName>.ini

If your Application is named "Sales", your `<ApplName>.ini` file is named SALES.ini.

The `<ApplName>.ini` file contains various settings that are in effect during the execution of your runtime Application. When the Application runs it reads the values in `<ApplName>.ini` and sets itself up accordingly. Some `<ApplName>.ini` file entries are designed to override settings you made in ACE. This allows you to alter these settings without recompiling the Application.

This section lists various `<ApplName>.ini` settings. The default settings are appropriate for most Applications. Commonly changed settings have an interface in the runtime environment. You should use this interface for those settings. It is accessed from the runtime environment menu *Options* > *General*.

Less commonly changed settings do not have an interface. These settings must be changed manually. You can edit the `<ApplName>.ini` file with any non-formatting text editor.

The Program Section of <ApplName>.ini

The following table describes selected settings in the `[Program]` section of `<ApplName>.ini`.

Table 53. Program section of the <ApplName>.ini (Sheet 1 of 4)

Listing in *.ini File	Description
LogoffType = 0	Sets the manner in which you exit your Application. "0" = prompts a dialog box with the warning: "You're about to Exit Application. Are you sure?" Then you disconnect from ACE but your system remains connected to the host computer. Any other entry prompts the logoff message enables an option to disconnect from the host or logoff. logoff=disconnect from ACE and logoff the host system. disconnect=disconnect from ACE but remain connected to the host.
TimeOutLimit=20000	Set the number of milliseconds that the ACE runtime waits for a response from the host.
DisplayAttributes=0	"0" = Do not show attributes in the full screen of the runtime Application. "1"= Show attributes in the full screen of the runtime Application.

Table 53. Program section of the <ApplName>.ini (Sheet 2 of 4)

Listing in *.ini File	Description
FirstSubApplName=	"*" = Begin your generated Application from any screen in the host application. "AAA" = Where "AAA" equals the name of the Subapplication. You can enter the name of a specific Subapplication that starts your generated Application. This setting overrides the setting in the Converter.
DisplayLineMsg=2	"0" = Do not display line messages "1" = Display messages in a window only. "2" = Display messages in the DIL only. "3" = Display the message in both the DIL and the message box.
HandleHostHelp=0	"0" = Do not enable host help "1" = Enable host help. When set, Help can be activated from the pull-down menu in the runtime window.
HandleMessageHelp=1	1= Prompts a button on the side of the DIL. The button activates a System-Triggered method, ***UserHostMethodHelpRequested***.
HandleAttention=0	"0" = Do not enable host attention "1" = Enable host attention. When set, Attention can be activated from the pull-down menu in the runtime window.
MessageHandlingAfter Default=1	Controls ACE default handling of methods and displays a message in the DIL prior to calling up a user defined method.
DIL=1	"0" = DIL is not displayed. "1" = DIL is displayed. This should always be set ON.

Table 53. Program section of the <ApplName>.ini (Sheet 3 of 4)

Listing in *.ini File	Description
HandleNextPrev Messages=1	Enables viewing multi-line host messages. Displays the Subapplication message window with + and - buttons for scrolling the message.
DDESupport=0	
ListZoom=1	“0” =Disable automatic editing of lists by double-clicking in runtime. “1” = List editing is performed in a zoom-in window. “2” =In-line list editing is performed on the list line.
HandleMsgsAfter Refresh=1	“0”=ACE displays messages before the window is displayed. “1”=ACE displays messages after the window is displayed.
RestoreMainWindow On Synchronize=1	“1”=Sets the system to synchronize after an unrecognized screen is displayed in the emulator, when the runtime window is minimized. Primarily used by OS/2 users.
MaximizeWindowOn Start=1	“0”= Your runtime application starts with the runtime window displayed. “1” =Your runtime application starts with the runtime window sized to its maximum. This flag can be set through the Application.
AutoStart=	“0”=Displays the runtime window and stops. “1”=Runs the Application automatically when the runtime is called up. This flag can be set through the Application.
AutoSynchronize=	Automatically synchronize on every change in the screen.

Table 53. Program section of the <ApplName>.ini (Sheet 4 of 4)

Listing in *.ini File	Description
AutomaticPopup Handling=1	"1"=Popup windows are recognized automatically. "0"=Popup windows are not recognized automatically.
WindowsPrefixChar=	Enter the character that indicates the *Windows* prefix for the accelerator key. The default character is the ampersand (&).

Entries Governing Prompt Controls

The application must use the same prompt keys and prompt values as used on the host. These are set in the following entries:

Table 54. Prompt keys and values (Sheet 1 of 2)

Listing in *.ini File	Description
[PromptKey]	Section Name
Global=<KeyName>	The Application sends the key named "KeyName" to the host as the prompt key, except for Subapplications that have a local prompt key set. *Example:* `Global=F4`
<Subname>=<KeyName>	When Subapplication "Subname" is displayed, the key named "KeyName" is sent to the host as the prompt key. There can be a different entry for each Subapplication. *Example:* `MBF004=F17`

Table 54. Prompt keys and values (Sheet 2 of 2)

Listing in *.ini File	Description
[PromptValue]	Section Name
Global=<Value>	The Application writes the value named “Value” in any prompted field, except in Subapplications that have a local prompt value set. Example: `Global=Chocolate`
<Subname>=<Value>	When Subapplication “Subname” is displayed, the Application writes the value named “Value” in any prompted field. Example: MBF004=Neapolitan To use a Subapplication specific value, set a Subapplication specific “KeyName” in the [PromptKey] section, even if the same key is used to prompt in that Subapplication. For example, if you want the F4 key to prompt Chocolate in all Subapplications, except MBF004 where F4 should prompt Neapolitan: [PromptKey] Global=F4 MBF004=F4 [PromptValue] Global=Chocolate MBF004=Neapolitan

Entries Governing Cursor Selection Menus

In some host applications, cursor selection menus require the cursor to be positioned in an edit field either at the beginning or the end of the line. In such cases, a particular value may need to be typed in the edit field before making the selection.

The entries in the following sections are used in these situations to define the location of the edit field and the value, if any, that must be placed in it.

Table 55. Entries governing cursor selection menus

Listing in *.ini File	Description
[MenuCursorSelection Key] **Global=(TAB/BACKTAB)** **<Subname>=(TAB/BACKTAB)**	Section Name *TAB* - Menu cursor selection menus have an edit field at the end of each line. *BACKTAB* - Menu cursor selection menus have an edit field at the beginning of each line. This setting can be overridden by a Subapplication specific setting: *TAB* - Menu cursor selection menus have an edit field at the end of each line. *BACKTAB* - Menu cursor selection menus have an edit field at the beginning of each line. Example: `MBF004=BACKTAB`

[MenuCursorValue]
 	
Global=<Value>
 	
<Subname>=<Value>
	Section Name
	
	The Application writes the value named “Value” in the edit field at the beginning or end of lines in a cursor selection menu, except in Subapplications that have a local menu cursor value set.
	Example:
	`Global=3`
	
	When Subapplication “Subname” is displayed, the Application writes the value named “Value” in the edit field at the beginning or end of lines in a cursor selection menu. Example:
	`MBF004=7`

The Emulator Section of <ApplName>.ini

The following table describes selected settings in the `[Emulator]` section of <ApplName>.ini

Table 56. Emulator section of the <ApplName>.ini (Sheet 1 of 3)

Listing in *.ini File	Description
WithWindowService= 1	"1" = Turns on additional services for emulators that support new versions of HLLAPI.
PanelsSaveDirectory=	Enter the directory where the panels should be saved in AutoLearn or when using the screen capture utility.
InputFile=	The name of the PNL file that will be displayed by the File emulator in the converter, when the emulator is activated.
EmulatorToView=	ACE/Connected (default is ACE) - Which emulator to show when viewing host screen on bleeding through. These are HLLAPI parameters that are only in effect when WithWindowServices=0 Rumba RC
EmulatorToViewFunc =	<HLLAPI Function Number> This function number should invoke the show function of the emulator.
EmulatorToViewStrin g=	<HLLAPI String Parameter> The above String Parameter.
EmulatorToViewLeng th=	<HLLAPI Length Parameter> The above Length Parameter.
EmulatorToViewRC=	<HLLAPI RC Parameter> The above RC Parameter

Table 56. Emulator section of the <ApplName>.ini (Sheet 2 of 3)

Listing in *.ini File	Description
HostColorTable=Host Colors	The color table saved in ACE is entered as the value. This can be set through the Application.
DisplayHiddenFields=1	1= Display the contents of hidden fields. 0=The contents of hidden fields remain hidden (default).
MainWindowSessionInfoPrefix	Used for multi-session environments. The value to be entered should be the name of another session. When using Alt=Tab the header should present the new session's name.
MinHostQuiet=nn	When the user sends data to the host (a S/390-type mainframe), the emulator automatically locks the keyboard. The keyboard remains locked until the host unlocks it. The server expects the host to unlock the keyboard after the final 3270-write operation which creates the screen.
	Sometimes the host may release the keyboard too soon, before sending the final message. To keep the server from trying to identify the host screen before is has received the complete screen, a timeout mechanism was developed for the JIS Server. This involves the `MinHostQuiet` setting (in the <ApplName>.ini file), which stalls the screen identification process for a specific length of time (specified in milliseconds), on the assumption that if the host has not sent anything for that time period then it has finished sending the screen.

Table 56. Emulator section of the <ApplName>.ini (Sheet 3 of 3)

Listing in *.ini File	Description
ContentionResolution=0|1	In the interest of providing the fastest possible response time, we now provide the option of synchronizing the unlocking of the user keyboard to the Send Data Indicator (SDI) sent by the host. If you use this option you can set `MinHostQuiet` to 0. To activate synchronization on the SDI, in the `[Emulator]` section of the `<ApplName>.ini` file, add the setting `ContentionResolution=1`. The Contention Resolution feature is not guaranteed to be effective in all environments. Application behavior with Contention Resolution set "on" should be evaluated carefully before making it a part of your production environment.

The Class Size Section of <ApplName>.ini

Table 57 describes selected settings in the `[Class Size]` section of <ApplName>.ini

Table 57. Class Size section of the <ApplName>.ini file

Listing in *.ini File	Description
String=400	Increases memory allocation for specific classes. This should remain as set unless specifically instructed to change this by JIS Interface Server Technical Support. Class size can be anywhere from 150 to 600.

The Window Layout Section of <ApplName>.ini

Table 58 describes selected settings in the `[Window Layout]` section of <ApplName>.ini

Table 58. Window Layout section of the <ApplName>.ini

Listing in *.ini File	Description
MainApplication=	The dimensions of the main window are saved here when SaveWindowLayout is turned on.

The Control Editor Section of <ApplName>.ini

Table 59 describes selected settings in the `[Control Editor]` section of <ApplName>.ini

Table 59. Control Editor section of the <ApplName>.ini

Listing in *.ini File	Description
EditWidthFactor=### **EditHeightFactor=###** **StaticWidthFactor=###** **StaticHeightFactor=###** **XSpacingBetween Controls=** **YSpacingBetween Controls=** **SpacingBetween Controls=**	Parameters for resizing controls. ### = % of change Example: If you enter 200 the size of the control will double.

The Fix <EmulatorName> Section of <ApplName>.ini

When the API fix is implemented, ACE creates a section in `<ApplName>.ini` called `[Fix <EmulatorName>]`.

Table 60 describes selected settings in the `[Fix <EmulatorName>]` section of `<ApplName>.ini`.

Table 60. Fix <EmulatorName> Section of the <ApplName>.ini (Sheet 1 of 3)

Listing in *.ini File	Description
TranslateNullsToBlanks=(0/1) **TranslateCharsToBlanks=** ***character to translate*** **TranslateChars=*character to translate***	Translate all ASCII nulls to blanks when accessing screen data.
ExplicitLength=(0/1)	Pass string length in one of 2 ways: - explicitly (0) as one of the parameters - implicitly (1) by terminating the string with a pre-chosen character. ACE operates in implicit mode however some emulators have bugs in implicit mode. Set this to force the use of explicit mode.
ScreenUpdateByUser=(0-4)	Control the screen update. 1= when emulator ignores HLLAPI typing. 2 = when emulator ignores typing in its own window. 3 = when emulator does not signal an update after HLLAPI typing. 4 = when emulator does not signal an update after typing in its own window as well.

Table 60. Fix <EmulatorName> Section of the <ApplName>.ini (Sheet 2 of 3)

Listing in *.ini File	Description
SwapRollKeys=(0/1)	When RollUp and RollDown keys operate in the opposite way as the PC's PageUp and PageDown keys.
Force24x80=(0/1)	Related to a bug in a version of Irma.
ProtectCurorShape=(0/1)	To retain the mouse-cursor shape.
IgnoreInvalidParameters=(0/1)	Clears up emulator error messages.
WinHLLAPISupport=(0/1)	Set to 1 for emulators that use the WinHLLAPI standard. You must set this flag before activating the emulator. Therefore, you have to manually create the `[Fix <EmulatorName>]` section in the <ApplName>.ini file and write this entry manually.
MultiExeSupport=(0/1)	
MultiExeHubDllName= ***name***	For VB Multi-Exe
MultiExeHubEntryName=*name*	For VB Multi-Exe
Trace=(0/1)	To trace or not to trace.
TraceFileName=*Filename*	The name of the file to contain the trace information.
TraceFileReset=(0/1)	Whether to reset the file contents for each new session.
TraceFixes=(0/1)	Whether to trace the call 'outside' the fixes as well (before the fixes upon entry, after the fixes upon exit).

Table 60. Fix <EmulatorName> Section of the <ApplName>.ini (Sheet 3 of 3)

Listing in *.ini File	Description
CalculateCursorPos=(0/1)	Set to “1” for emulators whose HLLAPI support for fixing cursor position is incorrect.

Converter.ini File

This section describes the settings in the Converter.ini file.

*.ini File Settings

Table 61 describes some of the settings in the Converter.ini file:

Table 61. Converter.ini file settings

Listing in *.ini File	Description
[Ignore Messages] **Window is positioned outside the boundaries=1**	Ignores 'VGA out of bound' messages.
[Ignore Messages] **10588=1** **10589=1**	Ignores “Pattern xxx not in section/msc” while loading the gkb by adding these values.
[Ignore Messages] **10581=1**	Ignores “Cannot match representation definition” message in batch.
[Ignore Messages] **10536=1**	Ignores “Cannot match decomposition definition” message in batch.
[Ignore Messages] **10533=1**	Ignores “Cannot match pattern definition” message in batch.

Entering Values for Windows Controls in the *.ini File

The procedure for changing values for *Windows* controls is as follows:

1. Set ACE to read the values from the *.ini file.
2. Enter the items in the *.ini files and set their format.
3. Enter new strings in the Specific.ini and <ApplName>.ini files.

This procedure is the same for all types of *Windows* controls. The following sections detail this procedure taking the combo box and check box controls as an example.

Set ACE to Read the Values From the *.ini File

To set ACE to read the values from the *.ini file:

1. In Design View, double-click on the control you wish to edit or double click on the control's name in the *Window Components* palette.

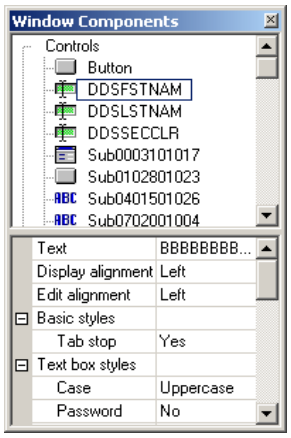


Figure 146. Window Components palette

2. The *Component* dialog box for the selected control opens.
3. In the *Component* dialog box, select the *Manager* tab.

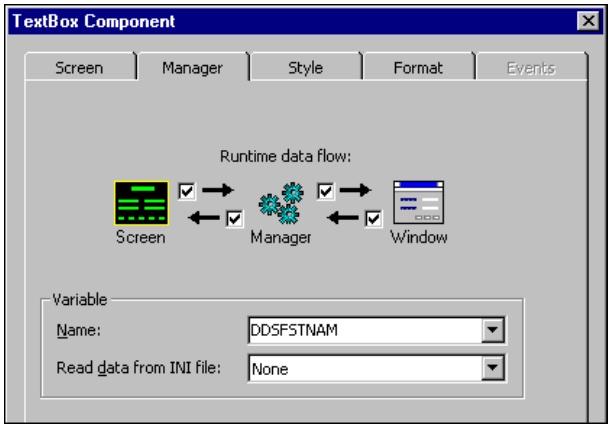


Figure 147. Manager tab in the Component dialog box

4 In the *Variable* section of the *Manager* tab set the *Name* and *Read data from INI file* options, where:

- *Name* is the name given to a window control.
  Note the name that the system has assigned to the combo box control. In this example the name of the combo box is DDSFSTNAM. To change this name, enter a new name in this dialog box.
- *Read data from INI file* sets ACE to read the parameters for the control from the Specific.INI file or the Specific.INI file cache section.
  In the *Read data from INI file* combobox, select Application or Subapplication, depending on whether the control is on the Application level (global) or for a specific Subapplication.

Table 62. Application and subapplication settings

Setting	Description
Application	The information is read from the `[program]` section in the<ApplName>.ini file. The information read applies to the entire Application.
SubApplicatio n	The information is read from the `[subapplication]` section in the <ApplName>.ini file. The information read is Subapplication-specific.

Updating a Combo Box from the *.ini File

A combo box is one of the objects used in the *Windows* interface for displaying a list of items for user selection. ACE offers two ways to enter items displayed in a combo box:

- By entering the items in the Representation Component dialog box in ACE. Changing these values requires recompiling the Application.
- By entering the items in the combobox.ini file and including this file in the Specific and Application *.ini cache. The system reads the entries from the *.ini file and updates the combo box automatically in runtime.

Entering Combo Box Items from the *.ini File

The procedure for creating dynamically changing combo box items is as follows:

1. In Design View set ACE to read the combo box items from the *.ini file as described previously in this chapter.
2. Enter the items in the combobox.ini file and include a line in the Specific.ini cache calling the combobox.ini.
3. Include a line for the combobox.ini file in the <ApplName>.ini file cache.

Note: The location of the entries in the *.ini file must be the same as your selection in the setup of ACE

Example 44. Entering combo box items from the *.ini file

The following example shows the entries for a combo box named DDSV5, and the purpose of each entry.

Table 63. Entries for example combo box and their purpose (Sheet 1 of 2)

Text in Combobox.ini File	Purpose
DDSV5_Flag=1	Set the flag to On
DDSV5=months	Enter the name of the entry containing the values for the combo box.
DDSV5_Format=months_Format	Enter the name of the entry containing the format.

Table 63. Entries for example combo box and their purpose (Sheet 2 of 2)

Text in Combobox.ini File	Purpose
months=January;January;February; March;April;May;	Enter the items to be listed in the combo box
months_Format=Jan;0;Feb;1; March;2;	Enter the format of the list

Note: The values in the <ApplName>.ini file are case sensitive.

1. Set the object flag to On

   The first line to be entered sets the flag for the combo box object to "1" which indicates that the format (specified in line 3) is to be read from the *.ini file. Enter the name of the combo box (as we retrieved it from the system in the procedure above). Then set the flag for this object to "1". The syntax is:

   Name_Flag=1

   Example: `DDSV5_Flag=1`

2. Enter the name of entry that contains the items to be displayed in the combo box

3. Create an entry in the *.ini file that contains the name of the combo box. The value for that combo box is the name of the item that contains the combo box entries.

   By creating a separate entry for the items in the combo box, the items can be shared by several objects in the Application. Therefore a separate entry is created for the objects to be listed in the combo box.

4. Enter the items to be displayed in the combo box. The item name is a logical name that is referenced by object. The items to be listed in the combo box must be separated by semicolons and the line must end with a semicolon. The item that precedes the first semicolon is the (optional) default item. The default item is listed first and is highlighted in the combo box.

   The syntax for this line is as follows:

   Name=[Pn];P1;P2;P3;

   Example: months=January;January;February;March;April;May;

Note: The default entry is listed twice. It is listed once before the first semicolon, indicating that it is the default, and it is listed again, as part of the list

The combo box will look as follows in runtime. The default is January.

Figure 148. Combo box during runtime

5 Set the format of the items to be displayed.

In this line the host screen and the *Windows* display are synchronized. Here you list the string in the host computer, followed by the item to be displayed in the window combo box, expressed by its placement in the line above. The items must be separated by semicolons and the line must end with a semicolon.

The syntax for this line is as follows:

```
months_Format=a;0;b;1;c;2;d;3;
```

For example:

months_Format=Jan;0;Feb;1;Mar;2;

In this line the string Jan from the host, corresponds to the first entry in the line above (0) which is January.

Note: The numbers must be sequential, and no numbers can be skipped.

Format Enhancement to Updating the ComboBox

Runtime performance can be improved, and the size of the *.ini file can be reduced by loading the values for the combo boxes into the system memory from the *.ini file, when the Application is called up. The size of the *.ini file is reduced by enabling logical references to a combo box format string. The format specified for the items in the combo box is defined once in the *.ini file, and referenced by other combo boxes using the same format.

In the *.ini file:

Assign a logical name to each of the formats to be used for the display of the items in a combo box. Create an item that loads these public formats into the system memory. These names and their values are loaded once into the system memory at startup. This saves time when displaying the Subapplication window.

Example 45. Format enhancement to updating the combo box

```
;DDSFieldDescription=NationalityFormat;months_format
;DDSFieldFormat=; ;0;AM;1;AU;2;BE;3;BR;
months_format=Jan;0;Feb;1;March;2;
```

Create one of two new methods:

To load format information after Application startup, create one of the following user-defined methods:

```
Domethod->Application-> UpdateAFormatHandler
DoMethod ->ComboBox->RefreshFormatHandler.
```

These two methods allow the user to refresh the format handler in case the *.ini file was changed during runtime.

Reading Check Box Values from the *.ini File

The GUI representation for Yes/No fields in the host system is usually a check box. The Yes/No field in the host can be expressed in a variety of different styles. Some host applications may use Yes/No, others Y/N and still others may use abbreviations in a foreign language.

ACE enables you to add a setting to your <ApplName>.ini file that will always interpret the format in the host correctly. The procedure for defining the format is as follows:

Note: This procedure is very similar to the format enhancement for combo boxes. The difference is that combo boxes contain a list of options, and check boxes are either checked or unchecked.

To define a format:

1. Set ACE to read the check box items from the Checkbox INI file and override the values in ACE.

   This is performed through the *Modify* function of the *CheckBox Component* dialog box in Design View. In the *Manager* tab you will find the name that was given to the check box. Change it if necessary, and set ACE to read the parameters for the check box from the INI file. See the ComboBox procedure described above for more details.
2. Enter the items in the Checkbox INI file and set the format of the checkbox.
3. Enter the following in the INI file to assign a logical name to each of the formats to be used by the check box and create an item that loads these public formats into the system memory. Set the flag for the checkbox to "1" (ON) and assign it the DDSField format.

Note: You must enter this in the section that you specified in the Representation Definition screen above

Example 46. Reading check box values from the *.ini file

Suppose we have a check box with the name "box1."

```
DDSFieldFormat=CheckBoxFormat;
CheckBoxFormat=;Y;0;N;1;
box1_Flag=1
box1_Format=CheckBoxFormat
```

The values for the check boxes are loaded into the system memory from the *.ini file when the Application is called up. This saves time when displaying the Subapplication window. Remember, you must include the checkbox.ini file in the cache of both the Specific.ini and the <ApplName>.ini files.

Saving the Last Check Box Setting

ACE can be set to save your last check box setting to the checkbox.ini file. When this is set, ACE reads the setting during runtime from the window, and saves it to the *.ini file. The next time the window is displayed it comes up with the setting last selected.

Set your system to save the value for a specific checkbox to the INI file, through its Representation Definition or in Modify/Representation mode. This setting is made in the *Manager* section of the *Representation Component* dialog box. See the procedure described above for full details.

When your system is set to save the data to the INI file, ACE saves your selection during runtime, and sets the checkbox to display your choice the next time the screen is displayed.

Creating a Logical Name for the Check Box

When ACE saves your check box setting, it uses the internal ACE name for your check box. You may find it advantageous to give your check box a logical name so that you can edit it, if necessary, in the *.ini file.

Supply a logical name to the GUI object in the *Manager* section of the *Representation Component* dialog box. Enter a name in the *Variable* field of the *Manager* section.

Translating Accelerator Keys

When you create your Application, you choose a language for the GUI. This choice automatically translates the names used for the system keys *Shift*, *Ctrl* and *Alt*.

Other keyboard keys must have their translations entered manually in the Application's <Language>.ini file.

The keys you can translate are:

Backspace, Caps Lock, Delete, Down, End, Enter, Esc, Home, Insert, Left, Num Lock, Page Up, Page Down, Pause, Right, Scroll Lock, Space, Tab, Up.

Part VII. About JI Integration Methods in JIS Interface Server

The JIS Interface Server and JI Integration products have complementary capabilities. On the one hand, JIS Interface Server enables you to extend and modernize legacy host systems. On the other hand, webMethod JI provides solutions for integrating legacy business systems. It is possible to integrate the capabilities of these two products, by invoking a webMethod JI method from an ACE method in the JIS Interface Server.

This document focuses on features and processes in JIS Interface Server, and does not intend to provide instruction on JI Integration. Information on JI Integration is supplied only to the extent necessary to understand how to work in ACE. For more detailed information on JI Integration, refer to the JI Integration documentation.

Chapter 20. Invoking JI Integration Methods from JIS Interface Server

JI Integration can generate Services and WSDL (Web Services Description Language) files that describe them. A Service contains one or more JI Integration methods. In JIS Interface Server, you can write an ACE method that invokes a JI Integration method, using the information in the WSDL file. This information is used to enable the ACE method writer to specify the input parameter values for invoking the JI Integration method, and the desired output parameter values.

In runtime, when the ACE method is activated, the JIS Server connects to the JI Integration Environment Manager. The Environment Manager activates the relevant Service and then invokes the JI Integration method. A JI Integration method can be invoked either in a synchronous mode in which the ACE method waits for a response, or in an asynchronous mode in which the ACE method does not wait for a response.

The following topics are presented in this chapter:

- When to Use This Feature
- Activation
- Workflow
- Design Time - Writing the ACE Method

When to Use This Feature

The ability to invoke a JI Integration method from an ACE method provides a wide range of capabilities, such as:

- Synchronous integration.

  Invoking a JI Integration method which returns output values to be used by the ACE method. For example, gathering information from screens without the need to convert them in ACE.
- Integration between different hosts.

  For example, a JIS Interface Server Application which extends a 3270 legacy application, and needs to retrieve data from a 5250 legacy application.
- Integration of web-enabled (3270 or 5250) applications with non-3270 or non-5250 systems that are supported by JI Integration. For example, integration of information from a VT application.
- Asynchronous integration.

Invoking a JI Integration method in the background. For example, invoking a JI Integration method that updates values to a database.

Note: Currently, only the Map-List-Map (MLM) data model in JI Integration is supported.

Software Requirements

The software requirements are:

- JI Integration Version 4.0 (or later).
  Note that "JI Integration 3.x Compatibility Mode" is also supported.
- JIS Interface Server Version 8.0A02 (or later).

When Not to Use This Feature…

It is important to know when not to use the integration feature. There are scenarios that are easily handled solely by JIS Interface Server, and do not require invoking a JI Integration method. For example:

- Sharing data between disparate screens to eliminate keystrokes. For example, using ACE methods to save the value of a field on one screen, in order to populate another field on a different screen in advance.
- Combining data from adjacent screens into a single GUI by using the JIS Interface Server Many-To-One feature.
- Controlling the flow of a JIS Interface Server Application. For example, creating ACE methods to skip to other areas in the Application.
- Client-side integration with other web sites, portals, etc.

Activation

The activation of the integration feature requires adding the following *.ini file entry:

```
[Integration]
IntegrateWithJacadaIntegrator=1
```

For ACE, add this entry to one of the ACE ini files.

For example, to activate the integration feature in a specific Application, add this entry to the Application's specific.ini file, located in the following directory:

```
<InstallDir>\appls\<ApplName>
```

To activate this feature in *all* your ACE Applications, add this entry to the guisys.ini file, located in one of the following directories:

```
<InstallDir>\KB_3270 or <InstallDir>\KB_400
```

For runtime, this entry is added automatically to the `<ApplName>.ini` file, as a result of runtime generation in ACE.

The runtime application *.ini file is `<ApplName>.ini`, and is located in the following directory:

```
<InstallDir>\appls\<ApplName>\RT32
```

Workflow

For a JIS Interface Server runtime Application to invoke a JI Integration method, a sequence of steps needs to be performed in both the JI Integration and JIS Interface Server products. These steps can be divided into three main phases:

- Design Time
- Runtime Generation
- Runtime

The following sections list the steps to be performed in each product, in each phase. Then, the document proceeds to describe each phase of the workflow.

Design Time

The following steps need to be performed in design time:

Table 64. Steps to be performed in design time

Step	Description
JI Integration	Create a JI Integration Service, which contains the method/s you would like to invoke from the JIS Server. Generate a WSDL file for this service.
JIS Interface Server	Write an ACE method that invokes the desired JI Integration method. For more information, see “Design Time - Writing the ACE Method” on page 315.

Runtime Generation

The following steps need to be performed in runtime generation:

Table 65. Steps to be performed in runtime generation

Step	Description
JI Integration	Generate the Service.
JIS Interface Server	Generate the runtime. In the *Generate Runtime* wizard, specify the JI Integration Environment Manager details.

Runtime

The following steps need to be performed in runtime:

Table 66. Steps to be performed in runtime

Step	Description
JI Integration	Deploy the Service.
JIS Interface Server	Configure any necessary runtime connection parameters. Deploy the runtime Application.

When an ACE method invokes a JI Integration method, the following occurs:

1. The JIS Server connects to the JI Integration Environment Manager using the JClient3 API.
2. The JIS Server requests a connection from the JI Integration Environment Manager to the required JI Integration Service.
3. The JIS Server uses the service connection to invoke the JI Integration method.

Design Time - Writing the ACE Method

This section describes how to write an ACE method that invokes a JI Integration method. It is assumed that a WSDL file has already been created in JI Integration.

In ACE, it is possible to write a method that performs the following in runtime:

1. Connects to a JI Integration Service.
2. Invokes a JI Integration method.
3. Parses the JI Integration method's outputs, when working in synchronous mode.

Writing the ACE method can be done using a wizard, which creates the necessary method lines automatically. It is also possible to maintain these automatically-generated method lines using a wizard. The JI Integration *Method Invocation* wizard is used for both these purposes.

In addition to the wizard, it is also possible to perform such tasks by writing individual method lines. For more information, see “DoMethods for Invoking a JI Integration Method” on page 321.

To help you understand the steps of the JI Integration *Method Invocation* wizard and the method lines it generates, some background information is provided. However, you can choose whether to skip directly to the description of the wizard, or read through the background information first.

The following topics are covered as background information for the wizard:

- JI Integration Map-List-Map Data Model
- Creating Structure and Array Objects in ACE Methods
- DoMethods for Invoking a JI Integration Method
- DoMethods that Operate on Structures and Arrays

JI Integration Map-List-Map Data Model

The integration feature supports the JI Integration Map-List-Map data model. In order to understand how to work in ACE with the input and output parameters of a JI Integration method, it is necessary to understand the Map-List-Map data model. This section provides an overview of this model.

The Map-List-Map data model is based on Java Map and Java List Interfaces, which are used to handle multi-level data structures in JI Integration.

A *Map* is a structure that contains key-value pairs.

Example 47. JI Integration Map-List-Map data model

This is a Map which describes a loan transaction:

Table 67. Transaction data

Key	Value
TransactionAmount	100.50
TransactionDate	01/01/01
TransactionType	Payment

A *List* is an indexed array of objects. An object in the List can be accessed using its index.

The Map-List-Map data model enables you to represent complex data structures.

A JI Integration method ***GetAllLoanInformation*** uses an account number as input, in order to retrieve the loan details and a list of loan transactions.

For each transaction the following information is available:

- Amount
- Date
- Type

Loan details include:

- AccountNumber
- BasePrincipal
- Name
- Term
- Collateral

Using the Map-List-Map data model, the outputs of the ***GetAllLoanInformation*** method can be represented as follows:

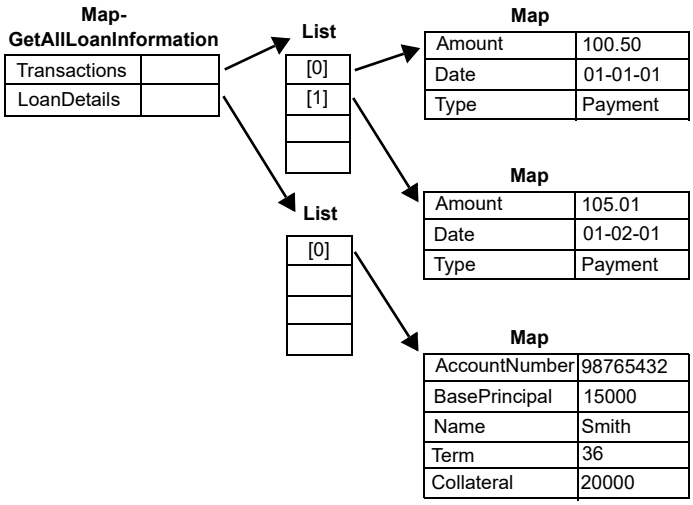


Figure 149. GetAllLoanInformation method

Creating Structure and Array Objects in ACE Methods

When writing an ACE method that invokes a JI Integration method, it is necessary to set the input parameter values and to select the output parameters whose values are to be returned to the ACE method.

For this purpose, there are two objects in ACE methods which correspond to the Map and List objects used in JI Integration:

- *Structure*

  A Structure in ACE represents the Map object used in JI Integration for input or output variables. Every object in a Structure has a string key and a value. The value is of one of the following types: a Structure, an Array or a String.
- *Array*

  An Array in ACE represents the List object used in JI Integration. An Array is an ordered list of objects, which are referenced using an index. The numbering of the index begins at zero. Each object in an Array is of one of the following types: a Structure, an Array or a String.

Therefore, the JI Integration Map-List-Map data model is represented in ACE by a Structure-Array-Structure data model.

Syntax for Specifying the Path of an Object

To specify the path of an object within a Structure-Array-Structure data structure, use the following syntax:

- To retrieve the value of a Structure object, use the name of the key.
- To retrieve an object from an Array, use its zero-based index enclosed in square brackets.
- Use the slash character ('/') as a separator.

This syntax can be used recursively, going as 'deep' as necessary.

Example 48. Syntax for specifying the path of an object

Using the ***GetAllLoanInformation*** example above:

- LoanDetails/[0]/AccountNumber - retrieves the account number string.
- Transactions/[1] - retrieves the structure containing the second transaction.
- Transactions/[1]/Date - retrieves the date string of the second transaction.

This syntax is used for specifying the 'path' parameter of ACE DoMethods that operate on Structures and Arrays. For more information, see "DoMethods Executed by Structures and Arrays" on page 323.

Method Line Types

The *Define Method* dialog box in ACE contains two method line types that enable you to create an Array or a Structure object:

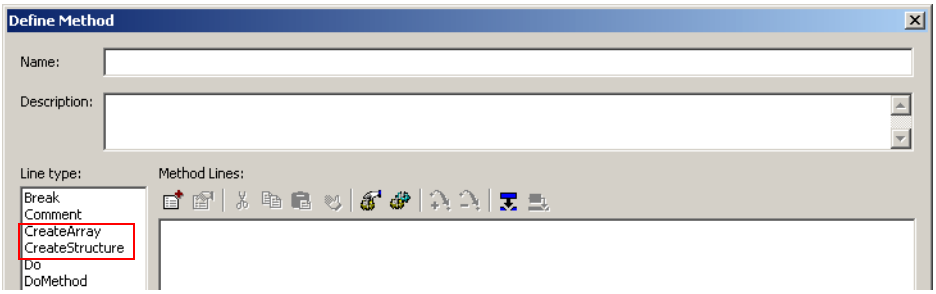


Figure 150. Define Method dialog box

CreateArray

The CreateArray method line type enables you to create an Array object, using the following dialog box:

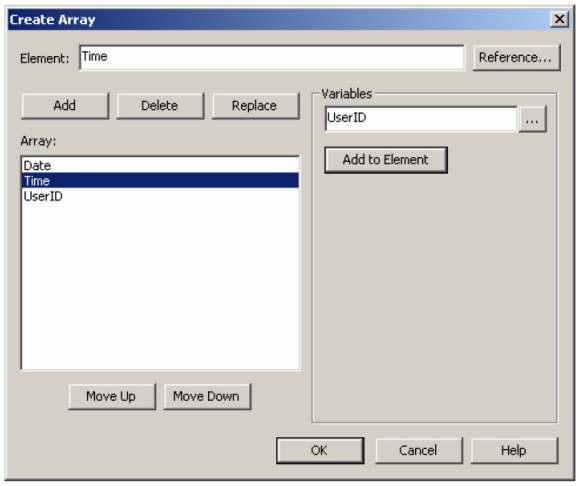


Figure 151. CreateArray dialog box

> **Note:** Strings must be surrounded by double quotes. In the dialog box above, all the Array elements are Variables

To create an Array:

1. Specify an *Element* of the Array in one of the following ways:
   By typing directly in the *Element* field; or
   By using the *Reference* button to reference an existing method line; or
   By selecting a *Variable* and then clicking the *Add to Element* button.

> **Note:** In a Current Subapplication method the Subapplication variables are listed in a list box.

2. Click the *Add* button to add the current Element to the *Array* list, or
   Click the *Replace* button to replace the selected element in the list with the current Element
3. Repeat steps 1-2 for each additional element in the Array.
4. Use the *Move Up*/*Move Down* buttons to order the elements in the Array.

Note: To delete an Array element, select it in the Array list and click the *Delete* button.

5 Click *OK* when the Array is complete.

In the above example, the following line is added to your method:

```
CreateArray: Array: '[Date, Time, UserID]'
```

Note: When using DDS Bridge, the *Create Array* dialog box contains some extra Indicator-related fields.

CreateStructure

The CreateStructure method line type enables you to create a Structure object, using the following dialog box:

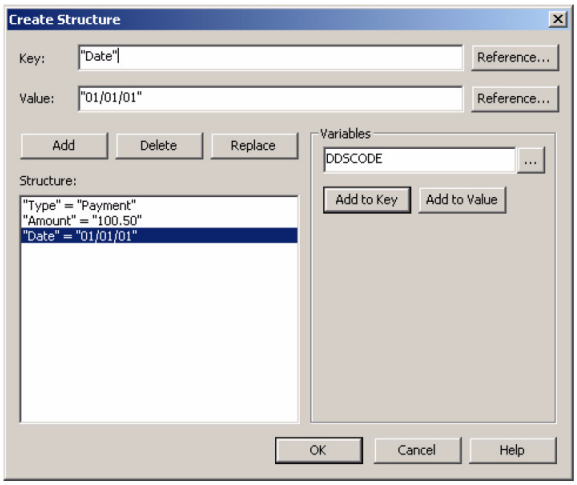


Figure 152. CreateStructure dialog box

Note: Strings must be surrounded by double quotes.

To create a Structure:

1 Specify a *Key* in one of the following ways:
By typing directly in the *Key* field; or
By using the *Reference* button to reference an existing method line; or
By selecting a *Variable* and then clicking the *Add to Key* button.

Note: In a Current Subapplication method the Subapplication variables are listed in a list box.

2. Specify a *Value* in one of the following ways:

   By typing directly in the *Value* field; or

   By using the *Reference* button to reference an existing method line; or

   By selecting a *Variable* and then clicking the *Add to Value* button.
3. Click the *Add* button to add a key-value pair to the Structure, using the contents of the Key and Value fields.

   Click the *Replace* button to replace the selected pair with the current key-value pair.
4. Repeat steps 1-3 for each additional key-value pair in the Structure.

Note: To delete a key-value pair, select it in the Structure list and click the *Delete* button.

5. Click *OK* when the Structure is complete.

In the above example, the following line is added to your method:

```
CreateStructure: Structure: '{"Type"="Payment", "Amount"="100.50",
"Date"="01/01/01"}'
```

Note: When using DDS Bridge, the *Create Structure* dialog box contains some extra Indicator-related fields.

DoMethods for Invoking a JI Integration Method

The following DoMethods are used when invoking a JI Integration method:

- ***ConnectToJIService***
- ***InvokeJIMethod***
- ***GetHostSessionName***
- ***CloseHostSession***

Table 68 through Table 71 describe each one of these methods.

Table 68. ConnectToJIService

Method Name	ConnectToJIService
Description	Connects to a JI Integration (JI) Service.
Executed by	ExternalData
Expected Return Type	JIService
Parameters	serviceName - the name of the service to connect to. hostSessionName - the name of the host session to use, when using a persistent, named session.

Table 69. InvokeJIMethod

Method Name	InvokeJIMethod
Description	Invokes a JI Integration (JI) method.
Executed by	JIService
Expected Return Type	Structure
Parameters	methodName - the name of the JI Integration method to invoke. inputStructure - the structure containing the method's input parameter values. waitForResponse - set to True to wait for the output of the JI Integration method (synchronous mode). Set to False to continue without waiting for the method output (asynchronous mode).

Table 70. GetHostSessionName

Method Name	GetHostSessionName
Description	Gets the name of the session on the host to which the JI Integration service is connected.
Executed by	JIService
Expected Return Type	String

Table 71. CloseHostSession

Method Name	CloseHostSession
Description	Closes the host session to which the JI Integration service is connected. This method should be used to close a persistent, named host session.
Executed by	JIService
Expected Return Type	Void

DoMethods Executed by Structures and Arrays

There are various DoMethods that can be executed by a Structure or an Array:

DoMethods for retrieving objects:

- ***GetArray***
- ***GetString***
- ***GetStructure***

DoMethods for operating on a Structure or an Array:

- ***ContainsObject***
- ***CopyObjects***
- ***RemoveObject***
- ***SetObject***

DoMethods related to Structure or Array properties:

- *Equals*
- *GetSize*

Table 72 throughTable 80 describe each one of these methods.

Table 72. GetArray

Method Name	GetArray
Description	Returns the Array located at the specified path.
Expected Return Type	Array
Parameters	path - the path from which to retrieve the Array.*

Table 73. GetString

Method Name	GetString
Description	Returns the String located at the specified path.
Expected Return Type	String
Parameters	path - the path from which to retrieve the String.*

Table 74. GetStructure

Method Name	GetStructure
Description	Returns the Structure located at the specified path.
Expected Return Type	Structure
Parameters	path - the path from which to retrieve the Structure.*

Table 75. ContainsObject

Method Name	ContainsObject
Description	Returns True when an object is located at the specified path.
Expected Return Type	Boolean
Parameters	path - the path to search in.*

Table 76. CopyObjects

Method Name	CopyObjects
Description	Copies the objects from the specified Structure or Array.
Expected Return Type	Void
Parameters	fromStructure - the Structure from which to copy the objects. -or- fromArray - the Array from which to copy the objects. *Note:* The type of this object must correspond to the type of the object executing this method - Structure or Array.

Table 77. RemoveObjects

Method Name	RemoveObjects
Description	Removes an object from the specified path.
Expected Return Type	Void
Parameters	path - the path from which to remove the object.

Table 78. SetObjects

Method Name	SetObjects
Description	Sets an object at the specified path.
Expected Return Type	Void
Parameters	path - the path into which to set the object.* object - the object to set at the specified path.

Table 79. Equals

Method Name	Equals
Description	Returns True if the objects are equal.
Expected Return Type	Boolean
Parameters	object - the object to compare to.

Table 80. GetSize

Method Name	GetSize
Description	Returns the size of the object.
Expected Return Type	Integer

Note: For information on the syntax of the path parameter, see "Syntax for Specifying the Path of an Object" on page 318.

Part VIII. About Customizing the ACE Menu Bar

It is possible to customize the ACE Menu Bar by adding items to it. This chapter explains how.

Chapter 21. Adding Items to the ACE Menu Bar

This chapter explains how to add additional operations of your choosing to the ACE menu bar.

Overview

ACE gives you the ability to add menu items to the ACE menu bar. Each menu item can display a custom menu structure of your choosing. Each item in the menu structure can be used to execute an external command.

Figure 153 shows the default appearance of the ACE menu bar.

Figure 153. The default ACE menu bar

Figure 154 shows an example of the ACE menu bar with two user items added to it. The items that the ACE administrator choose to add are “Source Safe” and “Editor”.

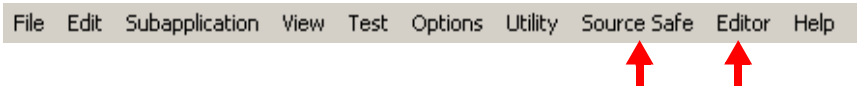


Figure 154. The ACE menu bar, modified

Figure 155 shows that the “Source Safe” item added to the menu bar is associated with a sub-menu. Source Safe is a code-management facility used at our hypothetical ACE administrator’s work site.

In our example, the Source Safe submenu added by the ACE administrator gives the ability to check-in the current subapplication, check-out the current subapplication, and to view the history of a subapplication, a library, and an application.

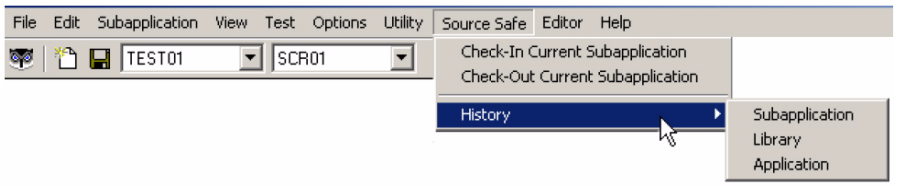


Figure 155. Example of a user defined menu structure

The ACE administrator can add a menu structure of his choosing to the ACE menu bar. It is the administrator's responsibility to make available the external programs executed by the menu items. The external programs can be conventional, commercially application programs (for example, Notepad, Source Safe, Calculator), or a batch file or program written in a language as Visual Basic, Java, C++ or C#.

Examples of what can be accomplished by a user entry on the ACE menu bar include:

- Check-in a subapplication to a source management tool.
- Check-out a subapplication from a source management tool.
- Edit java extensions.
- Compile java extensions.
- View the log from a JPack operation.
- Compile host code.
- Start the Monitor on the iSeries.

Each user item added to the menu bar must have at least one subordinate item. In Figure 155, for example, "Source Safe" is a user item added to the menu bar, and its subordinate items are "Check-In Subapplication", "Check-Out Subapplication", and "History". The subordinate items "Check-In Subapplication" and "Check-Out Subapplication" each invoke a specific program or command.

In Figure 155, "History" is an exception in that it does not point to a specific action. Rather it invokes a sub-menu, and each of the sub-menu items invokes a program or command.

Defining User Items on the ACE Menu Bar

To add user items to the ACE menu requires:

- The name and structure of the user items added to the ACE menu bar must be defined in a user-coded XML file that must be located in the JIS Interface Server install directory.
- Optionally, default settings for the ACE menu bar user items can be modified though *.ini file settings.

The XML File

An XML file must be created to define the items you wish to add to the ACE menu bar and the actions to be performed by each item. By default, ACE expects the file to be called `user_definitions.xml`. You can modify the default name via an *.ini file parameter, as explained beginning on page 339.

Here is an example of an XML file that adds a user item to the ACE menu bar.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <USER_DEFINITIONS>
   <MENUS>
      <MENU Name="IDK Main Menu">
            <POPUP Text="Source Safe">
                 <ITEM Text="checkin current subapplication"
                      Action="ci_subappl"/>
                 <ITEM Text="checkout current subapplication"
                      Action="co_subappl"/>
              <SEPARATOR/>

              <POPUP Text="History">
                   <ITEM Text="Subapplication" Action="his_subappl"/>
                   <ITEM Text="LIbrary" Action="his_lib"/>
                   <ITEM Text="Application" Action="his_appl"/>

              </POPUP>
          </POPUP>

         <POPUP Text="Editor">
              <ITEM Text="Notepad" Action="np"/>
              <ITEM Text="Codewright" Action="cw"/>
         </POPUP>
       </MENU>
    </MENUS>

   <ACTIONS>
      <ACTION Name="cw">
         <COMMAND
           Mode="NoWait">C:\WIN\util32\edit\CodeWright\cw32
         </COMMAND>
      </ACTION>

      <ACTION Name="np">
         <COMMAND Mode="NoWait">D:\WINNT\system32\notepad.EXE
         </COMMAND>
      </ACTION>

      <ACTION Name="ci_subappl">
         <COMMAND Mode="Wait">cmd /C c:\bat\checkin.bat
           %ApplicationName% %LibraryName%
           %SubapplicationName% </COMMAND>
```

```
      </ACTION>

      <ACTION Name="co_subappl">
         <COMMAND Mode="Wait">cmd /C c:\bat\checkout.bat
           %ApplicationName% %LibraryName%
           %SubapplicationName% </COMMAND>
      </ACTION>

      <ACTION Name="his_subappl">
         <COMMAND Mode="Wait">cmd /C
           c:\bat\history_subappl.bat %ApplicationName%
           %LibraryName% %SubapplicationName%
         </COMMAND>
      </ACTION>

      <ACTION Name="his_library">
         <COMMAND Mode="Wait">cmd /C c:\bat\history_library.bat
           %ApplicationName% %LibraryName% </COMMAND>
      </ACTION>

      <ACTION Name="his_appl">
        <COMMAND Mode="Wait">cmd /C c:\bat\history_appl.bat
          %ApplicationName%</COMMAND>
      </ACTION>
   </ACTIONS>

</USER_DEFINITIONS>
```

Figure 156 and Figure 157 show the two user menus produced by the sample `user_definitions.xml` file.

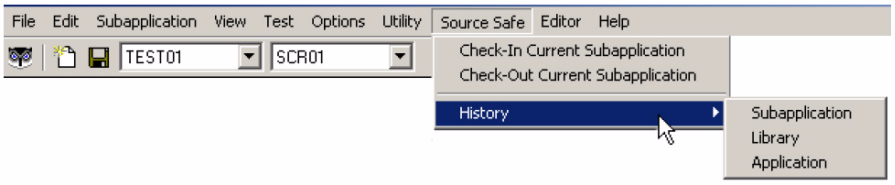


Figure 156. The example menu structures (1 of 2)

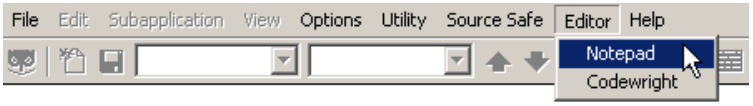


Figure 157. The example menu structures (2 of 2)

Tags in the user_definitions.xml file

Table 81 lists the tags available for use in the `user_definitions.xml` file when defining user menus on the ACE menu bar.

Table 81. Tags in the user_definitions.xml file (Sheet 1 of 3)

Element Name	Purpose
USER_DEFINITIONS	Highest level element ("root" element), brackets all of the MENUS and ACTIONS elements. (Required.)
MENUS	Contains all of the user menus. (Required.)
MENU	Defines a specific menu. In Figure 156 and Figure 157, the menu bar items `Source Safe` and `Editor` were produced by using the MENU element. (Your file must contain at least one MENU element.)
POPUP	Brackets a sub-menu item. (Your file must contain at least one POPUP element.) In Figure 157, the sub-menu which extends down from the `Source Safe` menu bar entry, as well as the sub-menu extending out from the `History` entry in the first sub-menu, are examples of POPUPs. To give a sub-menu item a popup, use nested POPUP statements. For an example of this, see the `History` sub-menu item and its definition.
Text attribute	Specifies the text of the ACE menu bar user entry. In a nested POPUP element, this specifies the text of a sub-menu item which produces a sub-sub-menu.
ITEM	Defines a lowest-level sub-menu item, one which executes an external program or command. You need at least one ITEM element.
Text attribute	Specifies the text of the sub-menu item.

Table 81. Tags in the user_definitions.xml file (Sheet 2 of 3)

Element Name	Purpose
Action attribute	Names the action to be taken when the menu item is selected. The action itself is defined in the named ACTION element.
SEPARATOR	Causes a horizontal separator line to appear on the submenu. Used to separate sub-menu items or groups of items. The SEPARATOR element is optional.
ACTIONS	Brackets all of the ACTION definitions.
ACTION	Specifies a single action to be executed. Brackets a COMMAND element.
Name attribute	The name of this action, referred to by the Action attribute of the ITEM element.
COMMAND	Specifies a program or command to be executed. You can specify a program or command directly. For example: `c:\myprog.exe` You can open a command window and run a program or command in the command window. For example: `cmd /C c:\mybatfil.bat` **Note:** If any part of the path you specify contains spaces, put the entire path in double quotation marks. For example: `“c:\Program Files\Tool Box\menu.exe”` Depending on your environment, this may not produce satisfactory results. In such a case, try using quotes just on the problematic nodes. For example: `c:\”Program Files”\”Tool Box”\menu.exe`

Table 81. Tags in the user_definitions.xml file (Sheet 3 of 3)

Element Name	Purpose
Mode attribute	Specifies how the program will be executed. Valid modes are: • **`NoWait`** - starts the program and immediately returns control to ACE, without waiting for the program to complete. • **`Wait`**- starts the program and waits for it to end before returning control to ACE. A dialog box that contains an "abort" option is opened. This gives the user the option to abort the requested program's execution. • **`Console`** - starts the requested program, opens a special console window, and directs the program's output messages (stdout and stderr messages) to a console window. Waits for the program to end before returning control to ACE.

Parameters for Use Within the ACTION Tag

Several predefined substitution parameters can be used in the ACTION tags in the `user_definitions.xml` file for specific variables whose values may not be known at the time of coding or are subject to change. These parameters are listed in Table 82. The following points apply:

- All parameters start and end with the "%" sign (percent sign).
- A double percent sign (%%) will be replaced with a single %.
- If the parameter has no value, the value "`_NONE_`" is returned.
- If the command processor encounters a parameter that is not defined, execution of the command halts. When possible, the processor displays a diagnostic message in the ACE window status bar.
- The parameters are not case-sensitive.

An example of the use of predefined parameters can be see in the `user_definitions.xml` file shown in Figure 156. In the ACTION named `co-subappl`, the associated COMMAND is:

```
<COMMAND Mode="Wait">cmd /C c:\bat\checkout.bat
%ApplicationName% %LibraryName% %SubapplicationName%
</COMMAND>
```

When the COMMAND is executed:

- `cmd` causes a Windows command window to be opened.
- `/C` is a parameter for the `cmd` command that says "execute the following command".
- the file at `c:\bat\checkout.bat` is executed.
- `%ApplicationName%, %LibraryName%, %SubapplicationName%,` and `c:\ACE\Exports` are input parameters for `checkout.bat`. In this particular example, the first three of these parameters (the ones bracketed by percent signs [%]) are predefined reserved words with specific meanings. Table 82 lists all of the predefined parameters.

Table 82. List of predefined parameters for use in user menus (Sheet 1 of 4)

Parameter	Description	Comment
%ApplicationININame%	The name of the so-called "specific.ini" file, taken from the parameter of this name in the `APPLS.INI` file.	
%ApplicationName%	Name of current open application.	
%ApplicationType%	The name of the particular "flavor" of JIS Interface Server under which the application was created. Value will be either "Jacada" or "Innovator" or "Studio".	
%HostApplicationName%	The name of the application to be connected.	For JIS Innovator and JIS Studio only.
%HostIP%	Host IP address or DNS name.	

Table 82. List of predefined parameters for use in user menus (Sheet 2 of 4)

Parameter	Description	Comment
%HostMonitorPort%	The port number of the monitor. Taken from the `<ApplName>.ini` file	For JIS Innovator and JIS Studio only.
%HostPort%	The host port number. Taken from the `<ApplName>.ini` file.	
%ImportExportDirectory%	Name of the export/ import directory.	
%JavaRootDirectory%	Path to the Java root directory.	
%JISRootDirectory%	The root directory of JIS Interface Server.	
%LastApplicationName%	Most recently opened application name.	If there is an application currently open, this parameter gives the same value as `%Application-Name%`.
%LibraryName%	Name of current open library.	
%LibraryType%	The name of the particular “flavor” of JIS Interface Server under which the library was created. Value will be either “JIS” or “Innovator” or “Studio”.	

Table 82. List of predefined parameters for use in user menus (Sheet 3 of 4)

Parameter	Description	Comment
%PackagesDirectory%	Name of the Jpacks directory.	
%ProgrammingEnvironment%	For JIS Innovator only. Returned value will be "JIS", "Innovator", or "Studio".	For non-Innovator systems, this parameter returns "`_NONE_`".
%RuntimeDirectory%	Name of the runtime directory.	
%RuntimeININame%	Name of the `<ApplName>.ini`.	
%RuntimeLanguage%	The language of the runtime. Value returned is: `Java`, `XHTML`, `HTML`, or `Visual Basic`.	
%ScreenImageFileName%	Name of the current subapplication screen image file.	
%ScreenImageName%	Name of the current subapplication screen image.	
%SubapplicationName%	Name of current open subapplication.	

Table 82. List of predefined parameters for use in user menus (Sheet 4 of 4)

Parameter	Description	Comment
%SubapplicationOrigin%	The name of the particular “flavor” of JIS Interface Server under which the subapplication was created. Value will be either “JIS” or “Innovator” or “Studio”.	
%SubapplicationRelationship%	Value returned is “Regular”, “Dependent”, or “Principal”.	

*.ini File Settings

There are certain default values associated with the use of user menus on the ACE menu bar. You can override default values associated with the user menus, through the use of *.ini file parameters.

Scope of *.ini File Settings

The particular *.ini file to which you add the parameters depends on the scope that you want for the parameter.

If you want the *.ini file settings to be effective for all applications under ACE, add the parameters to the `ace.ini` file (for mainframe hosts) or `ace400.ini` file (for iSeries hosts).

If you want the *.ini file settings to be effective only for a specific application, add the parameters to the `specific.ini` file belonging to the application in question. The `specific.ini` file for a given application can be found in the directory `<InstallDir>\APPLS\<ApplName>\`.

Parameters

First we will list the `jacadasv.ini` file parameters related to ACE user menus, then we will explain them.

```
[UserDefinitions]
```

```
DefinitionFiles=<file1><;file2><;file3>
Debug=0|1
LogFileName=file
```

Explanation

`[UserDefinitions]` All `jacadasv.ini` file parameters related to ACE user menus must be located in the UserDefinitions section of the INI file. Create a UserDefinition section by adding a line with the `[UserDefinitions]` header.

DefinitionFiles=<filepath1><;filepath2><;filepath3><;...> Use this line to specify the name of your user definition file(s). You can specify any number of user definition files. Specify the full path for each file. Do not code the pointy brackets (< >). If you specify more than a single file, separate the files with a semicolon.

For example:
`DefinitionFiles=c:\deffile_a.xml;c:\deffile_b.xml`

If this parameter is not used, ACE assumes there is a single definitions file and that its name is `user_definitions.xml`.

`Debug=0|1` You can cause the user menu commands to execute in "debug" mode by setting the Debug parameter to 1. The default value is 0, debugging off.

With debug mode set to 1 ("on"), when you execute an ACE user menu item a console opens automatically to display details about the interpretation and execution of the menu command. Debug mode can be helpful when developing a user menu item. You would not normally need to use debug mode once the user menu item works stably.

`LogFileName=<full path>` You can create a special log file that logs the invocation and execution of ACE user menu items. Use the `LogFileName` parameter to specify the full path of the log file. For example:

`LogFileName=c:\temp\usermenus.log`

If the `LogFileName` parameter is not used, no log file is created for ACE user menus.

Index

Symbols

.INI
- APIFix 297
- Class Size section 295
- Control Editor section 296
- Emulator section 293
- files 286
  - adding settings to applname.INI 305
  - and check boxes 305
  - controlling GUI objects from 137
  - converter INI file 299
  - creating dynamic combo box items 302
  - DDS 283
  - entering a DLL 63
  - entering values for Windows controls 300
  - error message 64
  - interpreting host Yes/No fields 306
  - loading files into cache memory 282
  - multiple files 281
  - multiple INI files 281
  - reducing size 304
  - saving data to 301
  - saving last check box setting 306
  - SDF 283
  - setting the object flag ON 303
  - updating a combo box from 302
- Fix EmulatorName section 297
- MenuCursorSelectionKey section 292
- MenuCursorSelectionValue section 292
- Program section 287
- PromptKey section 290
- PromptValue section 290
- runtime INI 286
- SDF INI 281, 283
- Window Layout section 296

A

Ampersand character support 198
Analyzing lists
- overview 215

Application components
- change management 27

Applications
- sharing 39

Array object 317
- creating 319
- DoMethods 323

B

Base year 61
Bleedthrough 191

C

Cache memory 282
- updating the cache in runtime 282

Caching
- host table pages 261

Calendar 52
Change management
- application tree structure 29
- considerations 37
- import and export operations 33

Check box
- formatting
  - interpreting 305
  - labeling check box 306
  - saving last check box setting 306
  - Yes/No fields 305

Class Size 295
Column
- splitting a column of data 237

Combo box
- default Item 303
- entering items from the INI file 302
- format enhancement 304
- updating from INI file 302

Conditionally display window 179
- subapplication type 179

Configuration management 27
- application components 27
- import/export directory 30
- principal and dependent subapps. 37

Control Editor 296
Controls
- alignment of 117

check box 306
combo box
updating from INI file 302
date
changing short date format 54
date calendar 51
editing controls 78
setting a uniform size for 107
Converter INI file 299
CreateArray 319
CreateStructure 320

D

Date 59
baseyear 61
control 51
changing short date format 54
formatting 56
localization 55
custom formatting 59
DDS
INI 283
Decomposition properties
expand button 257
Deleting
list headers and columns 234
Dependent subapplication 147, 149
change management 37
Dictionaries
creating 68
empty warning 68
entering a definition 69
invoking 68
overview 66
selecting the options 67
Display units 105
Dividing a list column and header 236
DLL for text format 63
DoMethods
and Jacada Integrator methods 321
CloseHostSession 321, 323
ConnectToJIService 321, 322
ContainsObject 325
CopyObjects 325
Equals 326
executed by structures and arrays 323
GetArray 324
GetHostSessionName 321, 323
GetSize 326
GetString 324
GetStructure 324
InvokeJIMethod 321, 322
RemoveObject 325
SetObject 326
ValidateDependentScreenByName 162
Dynamic
areas
with DDS and SDF files 137
sections 133

E

Emulator 293
Export operations 34
Exporting application components 27 to 38

F

Fix EmulatorName 297
Fixed list 260
Folded lists 241
Format dictionaries dialog box 68
Format handler
refreshing 305
Formatted items
assigning names 305
Formatting
after startup 305
text
Converter.INI file 63
Formfunc.dll 63
Function definition 99
accessing 99
Add Representation 124
Adjust Size by Text 110
advanced measurement 105
Align 117
Apply Window Layout 126
Center 115
Center in rectangle list box 115
Center Each 116
creating new 101
criteria 102
Equal Size type 107
Same As section 107

Equal Spacing 119
horizontal and vertical spacing 120
in rectangle 120
order section 120
spacing type section 119
Group of Functions 129
chosen function definitions list box 129
listing function definitions 100
Place 110
Distance section 104
Place Each 113
Place Each Relative to Display 114
Distance section 114
Set Font and Color 124
Spacing 123
Function description data box 66

G

GetToBottomOfList 252, 257
GetToTopOfList 252, 257

H

Handling lists 256
Header
splitting a column header 236
Headers
lists 225

I

Import operations 35
Importing application components 27 to 38
Integration
INI file entry 312

J

Jacada Integrator
integration with
activating 312
design time 313, 315
INI file entry 312
runtime 314
runtime generation 314
software requirements 312
when not to use 312
when to use 311
workflow 313
JI methods, invoking 313
design time 315
DoMethods 321
runtime 314
runtime generation 314
Map-List-Map data model 315
JClient3 314
JITGUI 191
setting subapplications as 191

L

Language of date control 55
Languages, multiple 307
Layouts
reading from INI files 81
List columns
deleting 234
splitting 237
List Extended Information 258
List Extended Information dialog box 261
List header 225
deleting 234
linking to columns 233
marking screen areas as 234
splitting 236
unlinking from columns 234
List object 316
List pattern definition type
representation definition 249
List with parameters 245
child patterns 219
list columns 230
placement 220
searched columns 223
ListColumns pattern definition 230, 249
Listing transitions 156, 157, 158
Lists 249
analyzing 215
design view 256
overview 249
fixed 260
folded lists 241
modifying
column type 239
ListColumn child patterns 239

paging mechanism 260
repeated columns list 243
pattern definition for 243
standard list 240
Localizing the date control 55

M

Many-to-One 147, 152, 153
analyzing dependents 154
analyzing the principal 152, 153, 154
creating the dependents 149
creating the principal 150
message handling 164
overview 147, 148
runtime 156
tabs 155
transition modes 157
transitions 156, 157, 158
UserMoveToDependentScreen method 158
ValidateDependentScreenByName 162, 163
Map object 315
Map-List-Map data model 315
MenuCursorSelectionKey 292
MenuCursorSelectionValue 292
Merging modifications 40
Message definition 201
types 199
Message handling
before/after window display 203
message help 204
message waiting indicator 204
methods 200
multi-line messages 203
Message handling methods
defining 201
modifying 202
Message help
configuring 204
Message waiting indicator 204
Method line types
CreateArray 318
CreateStructure 318
Methods
message handling 200
UserMoveToDependentScreen 158
MLM *see Map-List-Map data model*
Modes for table retrieval buffering 259
Multi-line edit fields
creating 195
Multi-line messages
configuration 203
Multi-line textbox fields
creating 195
Multi-page tables 261

N

Never Display Window 179
subapplication type 179
NO_ATTRS 193
Notification message 202

O

One To Many
methods 173
creating a method 173
One-To-Many
subwindows 168
adding a trigger to a subwindow 173
adding a trigger to the parent window 173
creating through the KnowledgeBase 172
creating using Add Control 168
displaying a hidden subwindow 173

P

Packing
an application 40
pack/unpack 39
PageDown 257
PageUp 256
Paging mechanism
lists 260
Parameters
button 68
Parms
unidentified 67
Path of an object 318
Pattern definitions
identifying in the analysis stage 64
list with parameters 245
ListColumn 230
Placement 220
Popup windows

batch mode 210
display in runtime 209
identifying in runtime 210
layouts 210
message handling 210
Principal subapplication 147, 150
change management 37
Program 287
PromptKey 290
PromptValue 290

R

Repeated columns list 243
pattern definitions 243
Representation component properties
style
table 251
Representation definition
lists 249
Retrieval buffering 259
all mode 259
cache mode 261
fixed mode 260
Runtime generation 314

S

Save data to INI File 301
Searched columns 223
Selection definition
accessing 83
by control type 87
by host location 88
associated with an area 90
controls specification 89
controls specification section 90
from primary pattern definition 90
location limitation 88
representing a pattern definition 90
by pattern definition 94
view button 94
by primary pattern definition 93
view button 93
by representation definition 94
by screen section 90
Fulfills all of the selection definitions 95
Fulfills at least one of the selection definitions 95
managing 84
Other than the selection definition 97
overview 83
types 86
workflow 87
writing new 85
Sharing subapplications 39
Skipping window-display in runtime
auxiliary methods 181
examples 183
overview 177
subapplication types 178
system-triggered methods 179
Splitting a list column 237
Splitting a list header 236
Standard list 240
Structure Object
DoMethods 323
Structure object 317
creating 320
Subapplication type
conditionally display window 179
JITGUI 191
Never Display Window 179
Subapplications
sharing 39
System
memory 304
System-triggered methods
GetToBottomOfList 252, 257
GetToTopOfList 252, 257
handling lists 256
PageDown 257
PageUp 256
UserMoveToDependentScreen 158
UserShouldWindowBeBuilt 180
UserSkipSubApplication 179

T

Table
style 249, 251
Table pages
caching 261
Table retrieval buffering 259
modes for 259
Tables

all mode 259
appearance and functionality of 249
changing column order 255
fixed mode 260
multi-page 261
page mode 259, 260
resizing rows and columns 255
Tabs
tab controls 141
create tab wizard 141
creating tabs 141, 142
deleting tabs 146
editing tabs 145
modifying tabs 144
moving tabs 146
renaming tabs 146
tab style options 145
Text
example data box 66
format definition dialog box 65
selections within 66
formatting
Available and Selected list boxes 65
Textbox
field 195
Transition modes 157
Translating the GUI 307

U

Unpack 39
Unpacking an application 41
Unpacking GKB files 45
UserMoveToDependentScreen method 158
UserShouldWindowBeBuilt 180
UserSkipSubApplication 179

V

ValidateDependentScreenByName 163

W

Window combo box
formatting in the INI 304
Window Layout section of INI file 296
Window layouts
adding instructions to 78
defining 73
organizing a definition 76
definitions
adding instructions 78
overview 71
window layout definitions 73
Windows controls
setting converter 300
updating 301
Wizards
create tab wizard 141

Y

Year 2000 61
Y2K 61