

# **JIS Interface Server: KnowledgeBase User's Guide**

Version 9.0

December 2024  
(originally released January 2005)

---

This document applies to JIS Interface Server Version 9.0 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 © 1992–2024 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-KB-UG-90-20241230**

---

# Table of Contents

<b>About this Guide</b> .....	<b>15</b>
Documentation Set .....	16
Document Conventions .....	17
Viewing the Documentation Online .....	18
 <b>Chapter 1. Explanation of Symbols</b> .....	 <b>19</b>
Table of Symbols .....	19
 <b>Chapter 2. KnowledgeBase Overview</b> .....	 <b>23</b>
What the KnowledgeBase Does .....	23
Modifying the KnowledgeBase .....	26
When to Modify the KnowledgeBase .....	26
Advantages of Local Modifications .....	27
Advantages of KnowledgeBase Modifications .....	27
Pieces of Information .....	28
Information Modularity .....	28
Making Use of the Modularity .....	29
Information and the KnowledgeBase: Primary Patterns .....	29
Modularity and the KnowledgeBase: Patterns .....	29
 <b>Chapter 3. KnowledgeBase Concepts</b> .....	 <b>31</b>
Information Concepts .....	31
Independence of Information .....	32
Completeness of Information .....	32
Complete Information and Partial Information .....	33
Information Independence .....	33
Information Completeness .....	33
Partial Information .....	33
Information and the KnowledgeBase .....	34
Information Sequences .....	34
Complete Information Sequences .....	34
Comparing Information Sequences by Content and Form .....	36
Partial Information Sequences .....	38
Complete and Partial Information Sequences .....	39
Conventional Structure Within the KnowledgeBase .....	39
Complete Information Sequences with Conventional Structure .....	40
Complete Information Sequences with Unconventional Structure .....	40
Character Groups Not Complete Information Sequences .....	41
Complete Sequences: Key to the KnowledgeBase .....	41
Variation in Partial Information Sequences .....	41
Informationally Identical Complete Information Sequences .....	42
Informational Analogous Complete Information Sequences .....	42

Information Sequences, Patterns, and Hierarchy .....	43
Patterns .....	44
Primary Patterns .....	44
Complete Information Sequences, Primary Patterns and Controls .....	44
Accommodating Variation in the ACE KnowledgeBase .....	46
A Detailed Example: Primary Pattern FKey03 .....	47
FKeyPrefix .....	48
FKeyTerminator .....	49
FKeyDescription .....	50
Number03 .....	50
 <b>Chapter 4. Modification Strategies .....</b>	<b>51</b>
Managing Choice-Elements Within Choice-Sets .....	51
Modifying a “OneOf” Type Choice-Set .....	51
Modifying a “Character Set” Type Choice-Set .....	52
The Advantage of Identical Character Sets .....	53
General Guidelines for Modifying Character Set Patterns .....	55
Modifying an “Iteration” Type Choice-Set .....	55
Case .....	56
New Pattern Definitions .....	59
Miscellaneous New Patterns .....	59
Priority Concepts .....	61
Using Priority to Modify the KnowledgeBase .....	63
Maintenance .....	65
 <b>Chapter 5. Selected Pattern Definitions .....</b>	<b>67</b>
Syntax and Conventions .....	67
The Pattern Name .....	67
Selected Pattern Definitions .....	69
AnyChar .....	69
CheckBox0or1Identifier .....	70
CheckBoxSeparator .....	71
CheckBoxViaINIField .....	71
CheckBoxYorNIdentifier .....	73
ColumnCheckBoxViaINI .....	73
ColumnComboBoxViaINI .....	74
ColumnHeaderTable .....	75
ColumnIgnore .....	76
ColumnInput .....	77
ColumnNumericInput .....	77
ColumnOutput .....	78
ComboBoxEntriesIdentifier .....	78
ComboBoxIdentifier .....	79
ComboBoxIdentifierCharacter .....	80
ComboBoxIdentifierChar0orMore .....	80
ComboBoxIdentifierFirstLine .....	81
ComboBoxIdentifierLine .....	82

ComboBoxViaINIField	82
ComboBoxWithIdentifierScattered	83
DefaultText	84
Digit_0_0to1	85
DynamicFKeyGroup	86
DynamicListCommandIteration	86
EndOfList	87
Entries	88
Entry	88
EntryPair	89
EntryPairs	90
FKey	91
FKey01	91
FKeyDescription	92
FKeyName	93
FKeyPrefix	94
FormatLeft	95
FormatRight	96
HeaderNoRD	97
ListCertainHeader	97
ListCommand	98
ListCommandDescription	99
ListCommandName	100
ListCommandNameChar	100
ListCommandTerminator	101
ListHeadersHardDelimiter	101
ListHeadersSoftDelimiter	102
ListLines	103
ListMoreIndicator	103
ListNoSelection	104
ListNotSubfile	105
ListWithoutHeader	105
Menu	106
MenuCommand	107
MenuDescription	108
MessageLine	109
Not_Attr	109
Number01	110
OutputCheckBoxViaINIField	111
OutputComboBoxIdentifierFirstLine	111
OutputComboBoxViaINIField	112
SystemIDOutputField	113
SystemIDViaSDFArea	113
SystemIDViaSDFField	114
SystemTimeOutputField	114
SystemTimeViaSDFArea	115
SystemTimeViaSDFField	116
UserIDOutputField	116

UserIDViaSDFArea .....	117
UserIDViaSDFField .....	117

---

## List of Figures

Three diagrams that illustrate the structure of Pattern Definitions . . . . .	19
A legacy host screen. . . . .	24
A graphical interface . . . . .	24
Dividing the host screen into sections . . . . .	25
Associating graphical controls . . . . .	25
Placing controls onto the graphical interface . . . . .	26
An independent piece of information. . . . .	33
Complete information sequences . . . . .	35
Host screen number one . . . . .	36
Host screen number two . . . . .	36
The difference between complete and partial sequences . . . . .	45
Primary pattern FKey03 . . . . .	47
FKeyPrefix . . . . .	48
FKeyTerminator . . . . .	49
FKeyDescription . . . . .	50
Number03 . . . . .	50
FKeyTerminator pattern definition tree . . . . .	52
BlankOrDotOrColon . . . . .	54
ComboBoxIdentifierChar character set . . . . .	54
FKeyTerminator . . . . .	55
Modified FKeyTerminator . . . . .	56
Title recognized by WindowCaptionFullScreen . . . . .	56
GUI window created by WindowCaptionFullScreen. . . . .	56
Setting location limits. . . . .	57
Host screen title on the second row . . . . .	57
Screen title in uppercase. . . . .	57
Screen title in uppercase and on second row . . . . .	58
Extending location parameter of WindowCaptionFullScreen . . . . .	58
WindowCaptionChar . . . . .	58
Modification of WindowCaptionChar character set . . . . .	58
Screen title on the first row . . . . .	59
Screen title on the second row . . . . .	59
WindowCaptionFullScreen_AdditionalText0to1 . . . . .	60
WindowCaptionFullScreen_AdditionalText0to1 location. . . . .	61
Section Pattern Definitions list box . . . . .	61
Representation component of FKey03 . . . . .	62
FieldsOneOf pattern . . . . .	62
Host screen pattern . . . . .	63
Result of analysis of FKey03 - menu item. . . . .	63
Result of analysis of FKey03 - button . . . . .	63
FKeySeparator . . . . .	64
FKeyWord_FKeySeparator1orMore . . . . .	64
New pattern using FKeyWord_FKeySeparator1orMore . . . . .	64

---

FKKey03Both . . . . .	65
Example of the pattern name . . . . .	67
AnyChar character set . . . . .	69
AnyChar . . . . .	69
CheckBox0or1Identifier . . . . .	70
CheckBoxSeparator . . . . .	71
CheckBoxViaINIField . . . . .	71
CheckBoxYorNIdentifier . . . . .	73
ColumnCheckBoxViaINI . . . . .	73
ColumnComboBoxViaINI . . . . .	74
ColumnHeaderTable . . . . .	75
ColumnIgnore . . . . .	76
ColumnInput . . . . .	77
ColumnNumericInput . . . . .	77
ColumnOutput . . . . .	78
ComboBoxEntriesIdentifier . . . . .	78
ComboBoxIdentifier . . . . .	79
ComboBoxIdentifierCharacter character set . . . . .	80
ComboBoxIdentifierChar0orMore . . . . .	80
ComboBoxIdentifierFirstLine . . . . .	81
ComboBoxIdentifierLine . . . . .	82
ComboBoxViaINIField . . . . .	82
ComboBoxWithIdentifierScattered . . . . .	83
DefaultText . . . . .	84
Digit_0_0to1 . . . . .	85
DynamicFKKeyGroup . . . . .	86
DynamicListCommandIteration . . . . .	86
EndOfList . . . . .	87
Entries . . . . .	88
Entry . . . . .	88
EntryPair . . . . .	89
EntryPairs . . . . .	90
FKKey . . . . .	91
FKKey01 . . . . .	91
FKKeyDescription . . . . .	92
FKKeyName . . . . .	93
FKKeyPrefix . . . . .	94
FormatLeft . . . . .	95
FormatRight . . . . .	96
HeaderNoRD . . . . .	97
ListCertainHeader . . . . .	97
ListCommand . . . . .	98
ListCommandDescription . . . . .	99
ListCommandName . . . . .	100
ListCommandNameChar . . . . .	100
ListCommandTerminator . . . . .	101
ListHeadersHardDelimiter . . . . .	101



---

ListHeadersSoftDelimiter. . . . .	102
ListLines. . . . .	103
ListMoreIndicator. . . . .	103
ListNoSelection . . . . .	104
ListNotSubfile . . . . .	105
ListWithoutHeader . . . . .	105
Menu. . . . .	106
MenuCommand . . . . .	107
MenuDescription . . . . .	108
MessageLine. . . . .	109
Not_Attr . . . . .	109
Number01 . . . . .	110
OutputCheckBoxViaINIField . . . . .	111
OutputComboBoxIdentifierFirstLine. . . . .	111
OutputComboBoxViaINIField. . . . .	112
SystemIDOutputField. . . . .	113
SystemIDViaSDFArea . . . . .	113
SystemIDViaSDFField. . . . .	114
SystemTimeOutputField . . . . .	114
SystemTimeViaSDFArea. . . . .	115
SystemTimeViaSDFField . . . . .	116
UserIDOutputField. . . . .	116
UserIDViaSDFArea . . . . .	117
UserIDViaSDFField. . . . .	117



---

## List of Tables

JIS Interface Server documentation set .....	16
Documentation conventions .....	17
Partial information sequence descriptions .....	38
Conveying identical information - partial sequence descriptions .....	42
Conveying similar information - partial sequence descriptions .....	43
Choice elements and choice sets .....	46
Choice elements and patterns for FKeyPrefix .....	48
Choice elements and patterns for FKeyTerminator .....	49



---

## List of Examples

Independence of information. . . . .	32
Complete information sequences - character group 'Menu' . . . . .	39
Complete information sequences with a conventional structure. . . . .	40
Complete information sequences with an unconventional structure. . . . .	40
Conveying identical information. . . . .	42
Conveying similar information . . . . .	43
The OneOf type choice-set . . . . .	52
The Character Set type choice-set . . . . .	53
Advantages of identical character sets. . . . .	53
BlankOrDotOrColon . . . . .	54
ComboBoxIdentifierChar . . . . .	54
Modifying an Iteration. . . . .	55
Creating a new Pattern Definition . . . . .	59
Order of FKey patterns in Section Pattern Definitions list box. . . . .	62
Order of Fields patterns in Section Pattern Definitions list box . . . . .	62



## About this Guide

---

This book explains what the KnowledgeBase is and describes techniques used to modify the KnowledgeBase. Specific strategies and concepts are presented here so that you can customize your KnowledgeBase and still preserve its modular design. This book also describes structural aspects of host screens and the GUI results they engender.

Understanding how the KnowledgeBase functions is important for only a few members of the conversion team. This is because of the global effect the KnowledgeBase can have on an entire application. Only those with knowledge about the legacy application and the targeted GUI design need to understand how the KnowledgeBase functions. Preferably these are people in managerial positions.

The KnowledgeBase information presented here is strictly conceptual in nature. Reading this book does not make you a KnowledgeBase expert but, it does give you enough information to become one with practice.

This book deals with the following topics:

- **Chapter 1 - "Explanation of Symbols"** explains the symbols used in the tree diagrams in this book.
- **Chapter 2 - "KnowledgeBase Overview"** shows you how ACE reads host screens by breaking them down into units. The KnowledgeBase uses these pieces of information to create a GUI according to pre-established rules. These rules called **Pattern Definitions** are at the heart of the KnowledgeBase and can be customized.
- **Chapter 3 - "KnowledgeBase Concepts"** explains how the KnowledgeBase interprets those units of screen information. Concepts such as information sequences, patterns and hierarchy are factors which impact the KnowledgeBase. Understanding these concepts and how to recognize them is important when strict parameters controlling the GUI design are in effect.
- **Chapter 4 - "Modification Strategies"** gets you working inside the KnowledgeBase. Here you see many screen captures that guide you through all the examples. While reading this chapter you should have ACE launched with an application open so that you can work along with the examples in the book. Emphasis in this chapter is placed on the concepts of Choice Elements and Choice Sets.
- **Chapter 5 - "Selected Pattern Definitions"** unveils how Pattern Definitions are built, what character sequences they recognize and if modifications are allowed.

## 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**

<b>This book...</b>	<b>Contains...</b>
<i>JIS Interface Server: Getting Started with the Automated Conversion Environment</i>	Startup information and an introduction to the Automated Conversion Environment (ACE).
<i>JIS Interface Server: Basic User's Guide</i>	Full explanations of the ACE Views and how to use them
<i>JIS Interface Server: Advanced Topics</i>	Explanations of advanced features that give your application extra functionality.
<i>JIS Interface Server: KnowledgeBase User's Guide</i>	In-depth information about the way the ACE KnowledgeBase is designed and how to work with it.
<i>JIS Interface Server: Java Client User's Guide</i>	Information for migrating your host application to Java.
<i>JIS Interface Server: XHTML Client User's Guide</i>	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 <b>once</b> . (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 <b>twice</b> . (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.
<i>italics</i>	Names of applications, programs, menus, dialog boxes, and libraries.
<b>Bold</b>	Menu options, and items, dialog boxes and items to be selected from a dialog box. The names of pull-down menus.
<b><i>Bold Italics</i></b>	Pattern definitions, representation definitions, message definitions, method names, layout names, section names, selection definitions, function definitions.
<b>BOLD + UPPERCASE</b>	Keyboard shortcuts: Press the <b>SHIFT</b> key. Press <b>CTRL + Z</b> .

## Viewing the Documentation Online

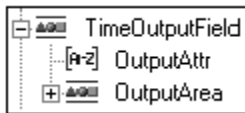
---

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

# Chapter 1. Explanation of Symbols





---

This book uses tree diagrams to illustrate the structure of Pattern Definitions:



**Figure 1. Three diagrams that illustrate the structure of Pattern Definitions**




In more detail:













- The pattern *TimeOutputField* contains the child patterns *OutputAttr* and *OutputArea*.
- The  symbol indicates that *OutputArea* contains child patterns, but that these child patterns are not displayed on the diagram.
- The  symbol indicates that all the child patterns of *TimeOutputField* are displayed on the diagram.
- The two  symbols indicate that *TimeOutputField* and *OutputArea* are Horizontal Group type patterns.
- The  symbol indicates that *OutputAttr* is a Character Set type pattern.


## Table of Symbols

---

The following table presents the meanings of each of the symbols used in the tree diagrams.

Symbol	Pattern Type
	Character Set
	Dynamic Group
	Dynamic Iteration

Symbol	Pattern Type
	Horizontal Group
	Horizontal Iteration
	List
	List Column
	List with Parameters
	One Of
	Popup Border
	Scattered Group
	String
	Vertical Group
	Vertical Iteration
	Child Patterns not displayed.

Symbol	Pattern Type
	Child patterns displayed.



## Chapter 2. KnowledgeBase Overview

---

The KnowledgeBase is a customizable tool in ACE which, when configured properly, recognizes all screens patterns in an application and automatically associates GUI representations to them. The key to making the most of the KnowledgeBase is modifying it to read your applications screens.

This chapter describes:

- What the KnowledgeBase does.
- How and when to modify the KnowledgeBase.
- How the KnowledgeBase reads screen information.
- Primary Patterns - the rules that control the KnowledgeBase.

ACE ships with many KnowledgeBase rules installed but, because legacy applications vary so widely there is a good chance your KnowledgeBase will need modifications to meet your applications particular needs. How and when you modify the KnowledgeBase depends on whether you want the effect to be local or global.

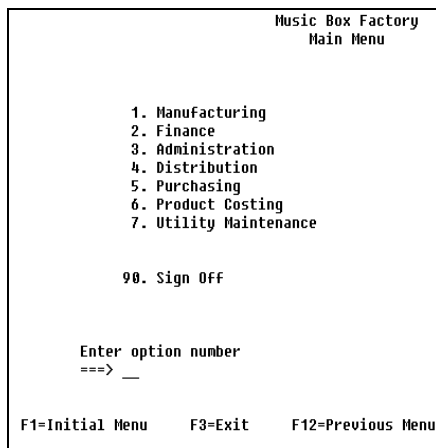
ACE works by breaking down the legacy screens into pieces of information. These pieces can be any combination of characters which the KnowledgeBase then reads. The rules that govern the KnowledgeBase in associating pieces of host information with GUI controls are called Pattern Definitions. Pattern Definitions are introduced here but, more detailed information on this topic is to come in subsequent chapters.

### What the KnowledgeBase Does

ACE is a multifunction tool that enhances your legacy application by adding a graphical interface consistent with current operating systems. Additionally ACE allows you to include added functionality without re-coding the legacy application. Added functionality allows your legacy application to integrate present technologies, such as the World Wide Web, intranets, Java, Visual Basic, as well as future technologies. ACE uses the KnowledgeBase when it:

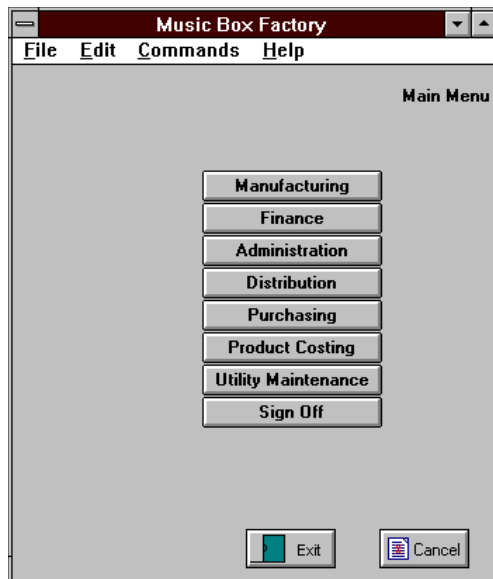
- Compares groups of host screen characters to KnowledgeBase Pattern Definitions and identifies which patterns are satisfied;
- Reads from the KnowledgeBase the Representation Components of an identified pattern and adds these components, both controls and methods, to the graphical interface. ACE

In order to accomplish this ACE must first perform the steps of converting a legacy host screen:



**Figure 2. A legacy host screen**

to a graphical interface:



**Figure 3. A graphical interface**

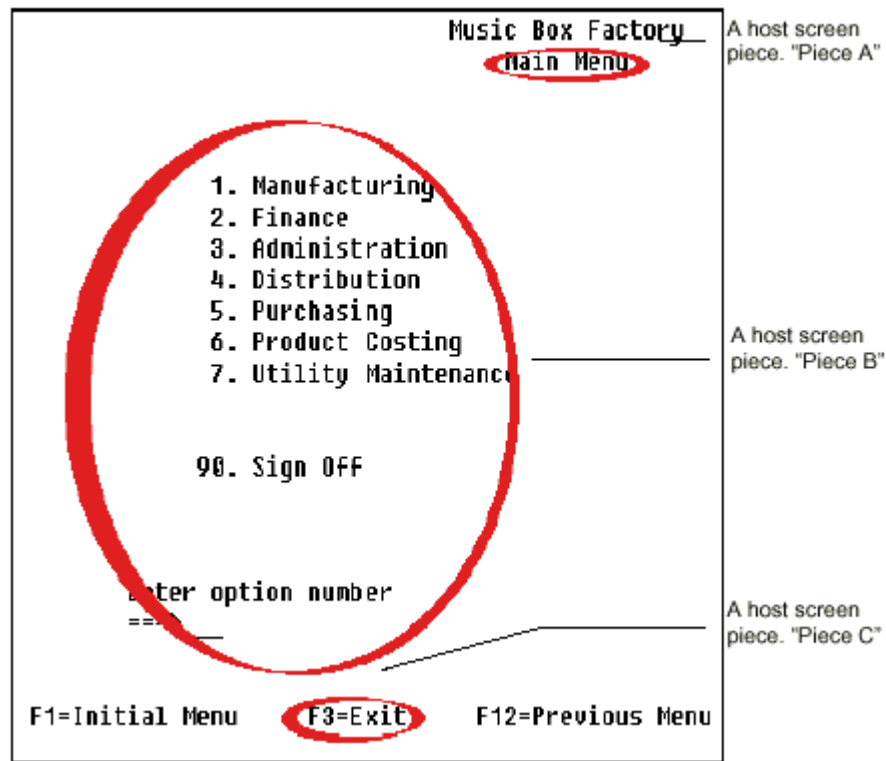
This is accomplished by the KnowledgeBase.

The KnowledgeBase contains the rules ACE follows when it converts host screens to graphical user interfaces. These rules determine which controls appear on the GUI, as well as their size, color and position.

ACE uses the KnowledgeBase rules to:

- 1 Divide a host screen into several sections:

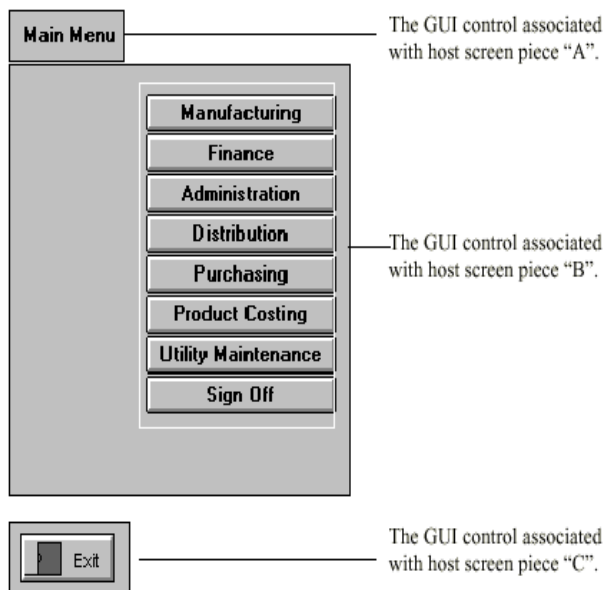




**Figure 4. Dividing the host screen into sections**

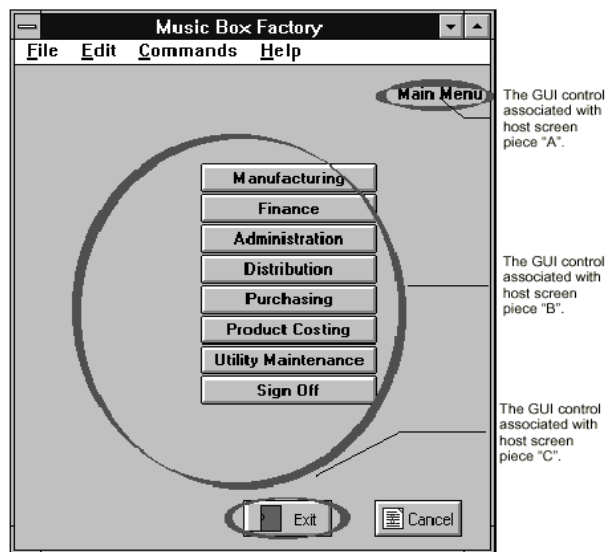
In this picture of a host screen, three of the host screen's pieces are circled.

- 2 Associate a graphic control with each host screen section, and determine the specifics, such as size and color, of each control:



**Figure 5. Associating graphical controls**

### 3 Place each control on the graphical interface.



**Figure 6. Placing controls onto the graphical interface**

## Modifying the KnowledgeBase

The ACE KnowledgeBase contains the rules ACE follows when it converts host screens to graphical interfaces. The ACE KnowledgeBase already contains a comprehensive set of rules when it ships. These rules reflect the standard interfaces of both host and graphical systems, and these rules allow ACE to provide basic functionality “out of the box.” Nevertheless, you will need to modify the KnowledgeBase to reflect the specific characteristics of your host application and to customize your application’s graphical interface.

## When to Modify the KnowledgeBase

You must “instruct” ACE on how to handle the specific characteristics of your application, but your instructions will not always be via the KnowledgeBase. In general, it is faster to make a single change locally than it is to modify the KnowledgeBase.

Therefore, when you need to make a change because of a situation present on a single screen, you will probably make the change locally using Design View. However, when the item you are changing is present in several host screens, it is more efficient to make the change via the KnowledgeBase.

## Advantages of Local Modifications

**Local modifications are usually faster than KnowledgeBase modifications.**

Modifying a Pattern Definition in the KnowledgeBase involves a few steps. Local modifications often require no more than placing a section boundary over an area of the screen.

**Local modifications are easier to fine tune.**

It is easier to change the position of a control, for instance, by dragging it than it is to modify the window layout. Visual composition is also easier with the window displayed.

**Some situations can only be handled via local modifications.**

There are some screen situations that can only be properly analyzed by a person who is familiar with the host application.

## Advantages of KnowledgeBase Modifications

**A modification that must be applied repeatedly is faster via the KnowledgeBase.**

It takes less time to modify the KnowledgeBase than to modify many screens.

**KnowledgeBase modifications help prevent mistakes and oversights.**

It can be very tedious to apply the same local modification to screen after screen. When you are making a modification that involves several steps, you only have to “get it right” once when you perform the modification via the KnowledgeBase.

**KnowledgeBase modifications ensure a standard look to the converted application.**

A KnowledgeBase modification will affect the analysis and presentation of more than just the specific screen situation that needed better treatment. The KnowledgeBase modification will also affect the analysis and presentation of similar screen situations. Thus a single KnowledgeBase modification ensures that similar screen situations have similar GUI representations.

**KnowledgeBase  
modifications ease  
maintenance.**

A KnowledgeBase modification becomes an automatic part of the conversion process.

A new person who starts to work on an ongoing conversion does not have to be instructed in all the local modifications.

If new screens are added to the application, KnowledgeBase modifications will automatically be applied to them.

If you decide to alter the modification you only need to do it once. You do not have to apply the alteration individually to each affected screen. Moreover, you do not have to examine each screen to see if the alteration applies to that screen.

## Pieces of Information

The KnowledgeBase rules divide the screen into pieces. These pieces are single pieces of complete, independent information. The division created by the KnowledgeBase rules is designed to be “natural.” The division is intended to reflect that different pieces of a host screen convey different pieces of information to you too.

## Information Modularity

Pieces of host screen information have a modular structure. A result of the modular structure is that host screen pieces that contain different information can be similar in certain aspects.

The characters:

`F3=Exit`

form a piece of host screen information. The information is that pressing the **F3** key moves the application to the previous logical screen.

A different piece of information is conveyed by the characters

`F1=Initial Menu.`

The information here is that pressing the **F1** key moves the application to its Initial Menu screen.

These different pieces of information are similar because they have the same modular structure; they are formed from characters representing the key name, a separator character and characters describing the function of the named key:

<b>Key Name Characters</b>	F3, F1
<b>Separator Character</b>	=
<b>Description Characters</b>	Exit, Initial Menu

Thus, despite the fact that these two sequences contain different information, they are similar because their structure is the same.

## **Making Use of the Modularity**

The modular structure of host screen information is very useful. It means that a piece of information can be identified via its structure no matter what the information is.

When you see characters representing a key-name, followed by a separator character, followed by additional words, you know that pressing the named key will have the effect described by the words. You may not understand the words, because you are not familiar with the application, or because the words are in a language you do not understand. However, the “structure” of the character sequence tells you that the sequence contains information about the effect of pressing a function key.

## **Information and the KnowledgeBase: Primary Patterns**

The rules that the KnowledgeBase uses to associate complete pieces of host information with GUI controls are called Primary Patterns.

The Primary Patterns are independent of each other. A Primary Pattern associates a particular piece of host information with a particular GUI control without affecting which Primary Pattern associates any other piece of host information with a GUI control.

## **Modularity and the KnowledgeBase: Patterns**

The KnowledgeBase Primary Patterns are built out of smaller patterns in a modular way. The modular construction of Primary Patterns out of smaller patterns is designed to reflect the modular structure of host information.

The ACE KnowledgeBase is designed to be modified in a way that preserves the modular construction of the Primary Patterns. Modifying the KnowledgeBase in this way:

- Preserves the independence of the Primary Patterns;
- Preserves the modular structure of the Primary Patterns.

The goal of this book is to teach you how to modify the KnowledgeBase in this “modular preserving” way. We strongly recommend that you modify the KnowledgeBase in this way such that the KnowledgeBase continues to reflect the modular structure of host information.

## Chapter 3. KnowledgeBase Concepts

---

This introduction describes information concepts and how they are applied to host screens. The term information concepts is used here to describe host screen characters and character sequencing. Furthermore, character and character sequencing will pre-determine the structure of your Pattern Definitions.

This chapter describes:

- Information concepts.
- Information sequences.
- Pattern Definitions.

The KnowledgeBase uses the host screen information to create the GUI. In the first pages of this chapter you learn how information on the host screen is read by the KnowledgeBase. Complete information sequences convey a single piece of information about the host system or about a host operation. Partial information sequences, when combined, make up complete information sequences but do not by themselves convey information.

Pattern Definition is a term that you will read throughout ACE. The KnowledgeBase contains patterns, which are rules for classifying host screen characters and character groups. You can create new Pattern Definitions in order to classify any host screen character.

The concepts in this chapter are presented to you in two ways. The first explanation describes these information concepts using examples from everyday English. The second explanation describes the information concepts using typical host screen character sequences.

### Information Concepts

---

Host applications present screen character information in many different ways. The KnowledgeBase uses the way this information is presented when creating the GUI. Information can be independent and stand alone, meaning it is not affected by other information. Whether or not the information is complete or partial also affects the workings of the KnowledgeBase.

## Independence of Information

---

The sentence: *“The pitcher threw the ball past the batter for a strike.”* describes an event at a baseball game. When you read this sentence you understand the event *itself* in its entirety. The sentence conveys information to you.

The information in the example sentence above is independent of any other sentences that precede or follow it. Other sentences *add* information but they do not affect the information in the sentence itself.

Here are two examples which both use the above example sentence. In each example, a second sentence works together with the above sentence to add information, but the information in the first sentence is unchanged.

### Example 1. Independence of information



*“The pitcher threw the ball past the batter for a strike. The crowd roared in approval.”*

The two sentences together tell you that the crowd favors the *pitching* team, but the information about the **event** is unchanged.

*“The pitcher threw the ball past the batter for a strike. The crowd groaned in dismay.”*

The two sentences together tell you that the crowd favors the *batting* team, but the information about the **event** is unchanged.

In both examples, the first sentence *“The pitcher ...”* conveys the **same** information to you about an event at a baseball game. In both examples the information is independent of the second sentence.

## Completeness of Information

The information contained in the sentence

*“The pitcher threw the ball past the batter for a strike.”*

is complete. There is no ambiguity about the event described in the sentence.

The words that make up this sentence **do not** contain complete information, although they appear to. In the above sentence the word *“pitcher”* contributes information about a player in a baseball game. In the following sentence the word *“pitcher”* contributes information about a container for liquids:

*“A pitcher of ice-cold lemonade.”*

The word *“pitcher”* is ambiguous when standing by itself. You have no way of knowing whether *“pitcher”* means a baseball player or a container for liquids. The single word *“pitcher”* *does not convey complete information.*



## Complete Information and Partial Information

---

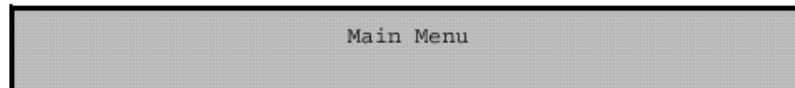
The word “*pitcher*” can contribute information when it is part of a sentence, even though by itself it is ambiguous. The word “*pitcher*” can be described as partial information.

The concepts of independent information, complete information and partial information are also applicable to host screen characters.

## Information Independence

---

The following piece of a host screen conveys an independent piece of information:



**Figure 7. An independent piece of information**

The characters and their position tell you that the current screen is a central takeoff point to other parts of the application. The information is independent of the menu items contained in the rest of the screen.

## Information Completeness

---

The information is also complete. The characters “Main Menu” located at the top of the screen are sufficient to convey the purpose of the current screen.

## Partial Information

---

The characters “Main Menu” by themselves are ambiguous. They do not necessarily convey information about the current location of the host application as they do in the example above. They might also contribute information about the function of a command key, as in the following character sequence:

“F5=Main Menu”

In this second example the characters “Main Menu” are a **part** of a complete piece of information. Together with the characters “F5” they convey that pressing the “F5” command key moves the application to the “Main Menu” screen.

In the first example, the characters “Main Menu” are a **part** of a complete piece of information. Together with their position at the top of a screen these characters convey that the current screen is the “main menu.”

## Information and the KnowledgeBase

---

The KnowledgeBase works by formalizing the concept of information. The KnowledgeBase contains rules that specify when a sequence of host screen characters form a complete and independent piece of information. The KnowledgeBase rules also specify which graphical element is associated with a complete piece of information.

The KnowledgeBase needs to be modified when it fails to identify a complete piece of information, or when it fails to associate a complete piece of information with the correct graphical element.

The remainder of this chapter gives full details of complete information and partial information, and then continues with how these information concepts are reflected in the KnowledgeBase rules. The following chapter, which explains modification strategies, tells you how to maintain the information structure of the KnowledgeBase when you modify the KnowledgeBase rules.

## Information Sequences

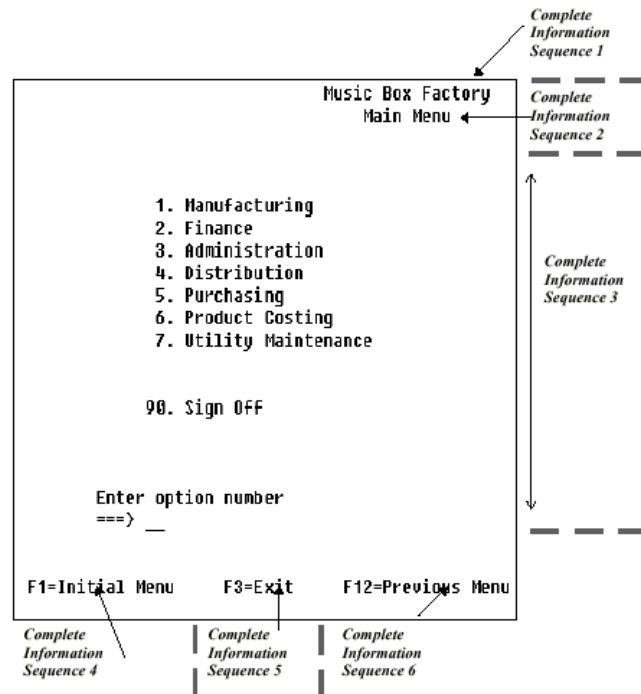
---

This section deals with the different types of Information Sequences.

### Complete Information Sequences

A **Complete Information Sequence** is a group of host screen characters that conveys to you a single piece of information about the host system or about a host operation you can perform.

In Figure 8, the host screen characters form six groups. Each group is a Complete Information Sequence:



**Figure 8. Complete information sequences**

#### Complete Information Sequence 1

Music Box Factory

The character group is a Complete Information Sequence that identifies the host application.

#### Complete Information Sequence 2

Main Menu

The character group is a Complete Information Sequence that identifies the current screen within the host application.

#### Complete Information Sequence 3

- 1 Manufacturing
- 2 Finance
- 3 Administration
- 4 Distribution
- 5 Purchasing
- 6 Product Costing
- 7 Utility Maintenance

## 8 Sign Off

Enter option number

===> \_\_\_\_\_

The character group is a Complete Information Sequence that prompts the task for choosing an option.

### Complete Information Sequence 4

F1=Initial Menu

The character group is a Complete Information Sequence that informs you that pressing the F1 key moves the application to the *Initial Menu* screen.

## Complete Information Sequence 5

F3=Exit

The character group is a Complete Information Sequence that informs you that pressing the **F3** key causes the application to move to the previous logical screen, according to the application's flow.

## Complete Information Sequence 6

F12=Previous Menu

The character group is a Complete Information Sequence that informs you that pressing the **F12** key moves the application to the previous menu.

## Comparing Information Sequences by Content and Form

Complete Information Sequences are informational units:

- All parts of a Complete Information Sequence are necessary to convey its information.
- A Complete Information Sequence is independent of other Complete Information Sequences.

In the two host screens:

F1=Initial Menu      F3=Exit

**Figure 9. Host screen number one**

and

[illegible]

**Figure 10. Host screen number two**

the information conveyed by “F3=Exit” is independent of the other characters on the screen.

Controls of the Graphical Interface correspond to Complete Information Sequences:

“F3=Exit” corresponds to: 

Different Complete Information Sequences can be:

- Informationally Identical, such as “F3=Exit” and “PF3 - Quit”  
These two Information Sequences convey identical information. Each informs you that pressing the third function key causes the host application to move to the previous logical screen.  
Informationally identical Information Sequences have identical Controls:

#### Sequence

#### Corresponds to:

F3=Exit



PF3 - Quit



- Informationally Analogous, such as “F3=Exit” and “F12=Previous Menu”

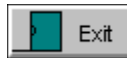
These two Information Sequences convey analogous information. Each informs you that pressing the specified function key causes the host application to perform the specified operation.

Informationally analogous Information Sequences have similar Controls:

#### Sequence

#### Corresponds to:

F3=Exit



F12=Previous Menu



- Informationally Unrelated, such as “F3=Exit” and “Music Box Factory”

These two Information Sequences convey different information. The first Sequence informs you that pressing the third function key causes the host

application to leave the current screen. The second Sequence tells you the name of the host application.

Informationally unrelated Information Sequences have unrelated Controls:

**Sequence****Corresponds to:**

F3=Exit



Music Box Factory



## Partial Information Sequences

Partial Information Sequences are the building blocks of Complete Information Sequences. A group of characters is a Complete Information Sequence when:

- It contains Partial Information Sequences whose individual meanings combine to form a unit of information.
- The Partial Information Sequences appear in a logical order

The character group “F3=Exit” is a Complete Information Sequence composed of four Partial Information Sequences. Each Partial Information Sequence has its individual meaning:

**Table 3. Partial information sequence descriptions**

Partial Information Sequence:	Description:
F	Identifies any function key.
3	Identifies the specific member of a given set.
=	Bridges identification and descriptive characters.
Exit	Describes the Complete Information Sequence.

The character group “F3=Exit” is recognizable as a Complete Information Sequence because:

- It is made up of Partial Information Sequences that combine to form a unit of information;
- The Partial Information Sequences appear in a logical order.

## Complete and Partial Information Sequences

The Partial Information Sequences that make up Complete Information Sequence have recognizable meanings but do not, by themselves, convey information:

The characters “Menu”

do **not** form a Complete Information Sequence; “Menu” by itself conveys no information. However, the character group “Menu” does have the meaning of a list of available options and it is therefore a Partial Information Sequence.

**Note:** It is important that “Menu” is not a Complete Information Sequence because we want a Complete Information Sequence to have a fixed GUI representation.

### Example 2. Complete information sequences - character group ‘Menu’



In Figure 8, the character group “Menu” occurs three times:

“Main Menu”

“F1=Initial Menu”

“F12=Previous Menu”

The character group “Menu” cannot have a fixed representation.

---

## Conventional Structure Within the KnowledgeBase

Complete Information Sequences have a conventional structure. The conventional structure is the type and order of the Partial Information Sequences that, *by convention*, make up the Complete Information Sequence.

It is necessary to use a “conventional structure” because a KnowledgeBase can only be finite and the number of possible Complete Information Sequences *that convey exactly the same information* is infinite. The conventional structure within the ACE KnowledgeBase reflects the most common forms of host Complete Information Sequences.

Examples of conventional and unconventional structure are given immediately below.

## Complete Information Sequences with Conventional Structure

Example 3 illustrates complete information sequences with a conventional structure.

### Example 3. Complete information sequences with a conventional structure

- ▶ The character group “F3=Exit” follows the conventional structure for a function key Complete Information Sequence:
  - It is made up of four Partial Information Sequences.  
(Refer to the example Example 2 on page 39 for their individual meanings.)
  - The Partial Information Sequences appear in a logical order.

## Complete Information Sequences with Unconventional Structure

Example 4 illustrates complete information sequences with an unconventional structure.

### Example 4. Complete information sequences with an unconventional structure

- ▶ The character group “F3=Exit” conveys the same information as “F3=Exit” despite the fact that it is missing the Partial Information Sequence “=”.  
“F3Exit” is a Complete Information Sequence with an unconventional structure.  
The character group “F3=Exit”  
conveys the same information as “F3=Exit” despite the fact that the Partial Information Sequences are in an unconventional order.  
“Exit=F3” is a Complete Information Sequence with an unconventional structure.

**Note:** Modifying the KnowledgeBase to handle the above situations automatically makes their unconventional structure conventional.



## Character Groups Not Complete Information Sequences

---

A character group that conveys no information, despite being similar to the conventional structure of some Complete Information Sequences, is **not** a Complete Information Sequence.

The character group “F10=” conveys no information. The character group is not a Complete Information Sequence.

## Complete Sequences: Key to the KnowledgeBase

---

A rule-based KnowledgeBase is possible because of the standardization that exists in both host and GUI systems.

There are two aspects to this standardization:

- Host Complete Information Sequences have conventional structures.
- Complete Information Sequences have conventional representations in GUI systems. Informationally equivalent Complete Information Sequences have identical representations.

The first of these points is the key to a KnowledgeBase. The conventional structure makes possible automatic recognition of host Complete Information Sequences.

Once the Complete Information Sequence is recognized, it is a straightforward process to add its conventional representation to the GUI. Similarly, the functionality of any associated task can be attached to the representation without difficulty.

## Variation in Partial Information Sequences

---

What is important about Complete Information Sequences is the type of information they convey and not the form in which the information is presented.

Therefore, the conventional structure of a Complete Information Sequence includes the *type* of meaning of the component Partial Information Sequences, but **not** the specific character groups nor their specific meanings.

Examples of different Partial Information Sequences with the same type of meanings appear in the following sections.

## Informationally Identical Complete Information Sequences

Example 5 illustrates complete information sequences who are informationally identical.

Two different Complete Information Sequences whose corresponding Partial Information Sequences have the same meaning are informationally identical.

### Example 5. Conveying identical information



The character groups “F3=Exit”, “Pf03 = exit” and “F3:Quit” are Complete Information Sequences that convey identical information, and they display conventional structure.

They are composed of different Partial Information Sequences, but the corresponding sequences have identical meanings:

**Table 4. Conveying identical information - partial sequence descriptions**

Partial Information Sequences:			Description
F	Pf	f	Identifies any function key.
3	03	3	Identifies the specific member of a given set.
=	/=/	:	Indicates termination of identification characters.
Exit	exit	Quit	Describes the Complete Information Sequence.

The character “v” represents the space obtained from the “spacebar.”

## Informational Analogous Complete Information Sequences

The conventional structure of a Complete Information Sequence is independent of the information it conveys. A Complete Information Sequence has conventional structure as long as its Partial Information Sequences have the appropriate *type* of meaning. Example 6 illustrates this.

**Example 6. Conveying similar information**

The Complete Information Sequences “F1=Initial Menu”, “F3=Exit” and “F12=Previous Menu” convey similar information. They are composed of different Partial Information Sequences, but the corresponding sequences have the same *type* of meaning:

**Table 5. Conveying similar information - partial sequence descriptions**

Partial Information Sequence:			Description:
F	F	F	Identifies any function key.
1	3	12	Identifies the specific member of a given set.
=	=	=	Indicates termination of identification characters.
Initial Menu	Exit	Previous Menu	Describes the Complete Information Sequence.

## Information Sequences, Patterns, and Hierarchy

Summary of Information Sequence Concepts:

- Character groups that convey information are called Complete Information Sequences.
- Complete Information Sequences that convey the same information have identical GUI representations.
- Complete Information Sequences that convey analogous information have similar GUI representations.
- Complete Information Sequences are composed of Partial Information Sequences whose individual meanings combine to form a unit of information.
- Complete Information Sequences have a conventional structure.
- The Partial Information Sequences that make up a conventional structure Complete Information Sequence do not have to have a specific meaning, but they must have a specific type of meaning.

## Patterns

The structure of the ACE KnowledgeBase is similar to the structure of Complete Information Sequences.

The KnowledgeBase contains *Patterns*, which are rules for classifying host screen characters and character groups. A given group of host screen characters may or may not satisfy the rules of one or more KnowledgeBase Patterns.

### Primary Patterns

The KnowledgeBase classifies host character groups by the information they convey. The Patterns it uses to do this are called Primary Patterns.

The Primary Patterns in the KnowledgeBase are designed to meet four requirements:

- 1 All Complete Information Sequences that convey the same information satisfy the same Primary Pattern;
- 2 Complete Information Sequences that convey analogous information **may or may not** satisfy the same Primary Pattern;
- 3 Complete Information Sequences that convey completely different information satisfy different Primary Patterns;
- 4 All the Complete Information Sequences that satisfy a given Primary Pattern convey the same or analogous information.

ACE creates the graphical interface from the Primary Patterns. Generally, each Primary Pattern is associated with one or more controls on the graphical interface.

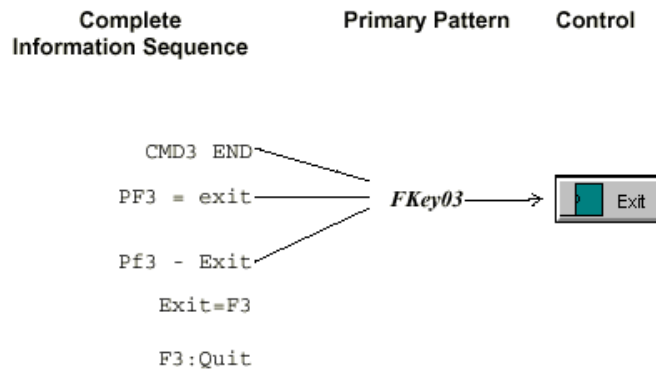
## Complete Information Sequences, Primary Patterns and Controls

A Complete Information Sequence is a specific group of host screen characters that conveys information.

**Note:** Primary Patterns are organization formats for groups of host screen characters. A Primary Pattern is not a Complete Information Sequence.

A group of host screen characters conveys information, and is therefore a Complete Information Sequence, whether or not there is a KnowledgeBase Primary Pattern that recognizes the sequence.

Figure 11, taken from the ACE KnowledgeBase illustrates the difference between Complete Information Sequences and Primary Patterns:



**Figure 11. The difference between complete and partial sequences**

Each of the five character groups listed above indicates that pressing the **F3** key causes the host to quit the current screen of the application. Therefore, all five character groups are Complete Information Sequences. Moreover, all five convey the same information.

However, only the first three character groups satisfy the KnowledgeBase Primary Pattern *FKey03*. The GUI control that represents *FKey03* is a “clickable” button.

The character groups “Exit=F3” and “F3:Quit” convey the same information as the other three groups, but they do not satisfy the Primary Pattern *FKey03*.

As all five of the above character groups convey the same information, all five should satisfy the same Primary Pattern. You modify the KnowledgeBase such that character sequences conveying the same information all satisfy the same Primary Pattern.

## Accommodating Variation in the ACE KnowledgeBase

The ACE KnowledgeBase is designed so that different variations of a Complete Information Sequence are recognized by the same Primary Pattern Definition.

The KnowledgeBase recognizes variations by means of Choice-Sets and Choice-Elements:

- A **Choice-Set** is a pattern which identifies the *type* of meaning a Partial Information Sequence should have as part of a particular Complete Information Sequence.  
“OneOf”, “Character Set” and “Iteration” are Choice-Set patterns. Each of these kinds of patterns indicates that any one of a group of different Partial Information Sequences can be part of the Complete Information Sequence.
- A **Choice-Element** is a *specific* Partial Information Sequence that can be part of the Complete Information Sequence.

**Table 6. Choice elements and choice sets**

CHOICE ELEMENTS		CHOICE-SETS
Pattern Type	Pattern Name	
OneOf	MenuSeparator	/
		.√
		.
Character Set	BlankOrDotOrColon	/
		.
		:
Horizontal Iteration	Blank1to2	/
		//

- The character “√” represents the space produced by pressing the Spacebar key.
- “OneOf” type pattern *MenuSeparator* is a Choice-Set. A Primary Pattern containing *MenuSeparator* can be satisfied by a character group with either “√” or “.√” or “.” in the corresponding position.
- “Character Set” type pattern *BlankOrDotOrColon* is a Choice-Set. A Primary Pattern containing *BlankOrDotOrColon* can be satisfied by a character group with either “√” or “.” or “:” in the corresponding position.
- “Horizontal Iteration” type pattern *Blank1or2* is a Choice-Set. A Primary Pattern containing *Blank1or2* can be satisfied by a character group with either “√” or “√√” in the corresponding position.

## A Detailed Example: Primary Pattern FKey03



**Figure 12. Primary pattern FKey03**

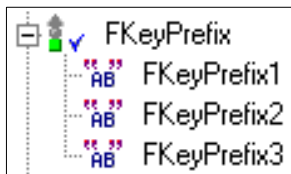
Primary Pattern **FKey03** is composed of four Choice-Set patterns: **FKeyPrefix**, **Number03**, **FKeyTerminator**, and **FKeyDescription**, with **FKeyPrefix** and **Number03** grouped together for reasons described below. These four patterns correspond to the four Partial Information Sequences described in Table 3 on page 38.

Thus, a Complete Information Sequence that satisfies **FKey03** must be a concatenation of four Partial Information Sequences. Each of the four Partial Information Sequences must be one of the Choice-Elements associated with the corresponding Choice-Set.

**Note:** The pattern **FKey03Name** is present because the “Accelerator” Representation Component needs to use text taken from both the **FKeyPrefix** and **Number03** patterns, and text must be taken from a single Pattern Definition.

The following subsections list the Choice-Elements associated with each of the four Choice-Sets.

## FKeyPrefix



**Figure 13. FKeyPrefix**

Choice-Set *FKeyPrefix* is satisfied by any one of thirteen Choice-Elements:

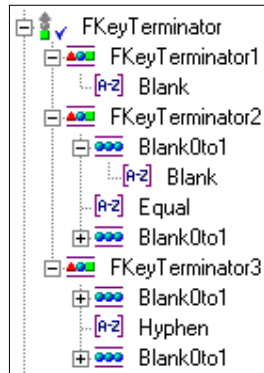
**Table 7. Choice elements and patterns for FKeyPrefix**

Choice Elements	Pattern
<b>F</b>	FKeyPrefix1
<b>CMD, CMd, CmD, Cmd, cMD, cMd, cmD, cmd</b>	FKeyPrefix2
<b>PF, Pf, pF, pf</b>	FKeyPrefix3

Note that the patterns *FKeyPrefix2* and *FKeyPrefix3* have several character groups associated with them because of the “ignore case” option in their Pattern Definitions.



## FKeyTerminator



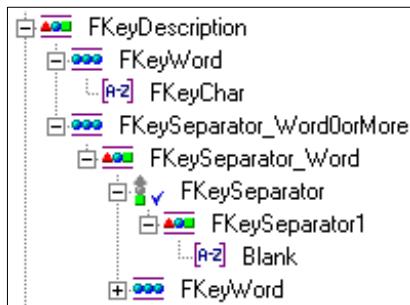
**Figure 14. FKeyTerminator**

Choice-Set *FKeyTerminator* is satisfied by any one of nine Choice-Elements:

**Table 8. Choice elements and patterns for FKeyTerminator**

Choice Elements	Pattern
/	FKeyTerminator1
=, /, =/, /	FKeyTerminator2
-, /, -/, /	FKeyTerminator3

## FKeyDescription



**Figure 15. FKeyDescription**

Choice-Set *FKeyDescription* is satisfied by any one of an essentially infinite number of Choice-Elements:

FKeyWord  
*FKeyWord*∨*FKeyWord*  
*FKeyWord*∨*FKeyWord*∨*FKeyWord*  
 ...

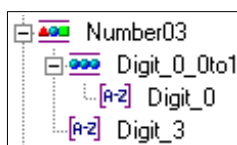
Each occurrence of *FKeyWord* is itself satisfied by character groups of the form:

Abcd  
 AAbcDe  
 Abcd/dfdD  
 ...

Thus, *FKeyDescription* is satisfied by an infinite number of Choice-Elements of the form:

Abcd  
 AAbcDevAbcd  
 Abcd/dfdDvAbcd

## Number03



**Figure 16. Number03**

Choice-Set *Number03* is satisfied by two Choice-Elements: 3 and 03

## Chapter 4. Modification Strategies

---

With this chapter the gloves come off and you begin working right inside the KnowledgeBase. All the procedures here are performed from within the KnowledgeBase Definitions window. Full details on individual steps are found in *JIS Interface Server: Basic User's Guide*.

This chapter describes:

- Managing Choice-Elements Within Choice-Sets
- Modifying a “OneOf” Type Choice-Set
- Modifying a “Character Set” Type Choice-Set Set
- Modifying an “Iteration” Type Choice-Set
- New Pattern Definitions
- Miscellaneous New Patterns
- Priority Concepts
- Maintenance

Learning how to manage Choice-Elements within Choice-Sets is important when maintaining the modularity of the KnowledgeBase. This chapter teaches you how to modify one of character sets and iteration type pattern definitions.

You will also learn how you can use pattern definition location parameters and the priority of pattern definitions to refine the host screen analysis. In this way you make the analysis more sophisticated.

Different techniques for tweaking pattern definitions are introduced. Tips such as using the letter case, priority and position of host screen characters help you achieve a high level of control over your KnowledgeBase.

### Managing Choice-Elements Within Choice-Sets

---

The preferred modification strategy is managing the Choice-Elements associated with a Choice-Set.

#### Modifying a “OneOf” Type Choice-Set

“OneOf” patterns are the entry point for the easiest kind of KnowledgeBase modification. When a “OneOf” pattern is not satisfied by a particular character group, you can create a new pattern that does recognize the character group and add the new pattern as a child pattern of the “OneOf” pattern.

**Example 7. The OneOf type choice-set**

"F3:Quit"

The character group above does not satisfy the Primary Pattern *FKey03* because the character group ":" is not one of the child patterns of "OneOf" pattern *FKeyTerminator*. The best solution to this situation is to make ":" a child pattern of *FKeyTerminator*.

To make ":" a child pattern of *FKeyTerminator*:

- 1 Create a Horizontal Group pattern *FKeyTerminator4* containing the Character Set pattern *Colon*.
- 2 Add *FKeyTerminator4* to the list of child patterns of *FKeyTerminator*.

The Pattern Definition tree for *FKeyTerminator* appears as follows:



**Figure 17. FKeyTerminator pattern definition tree**

**Make the new child pattern the last pattern listed.** ACE tries to apply child patterns in the order which they are listed. When you make the new child pattern the last one listed, you ensure that the new child pattern will not interfere with other character groups already recognized by the "OneOf" pattern.

**Note:** The ACE KnowledgeBase is designed for modification via "OneOf" patterns. This is why there are "OneOf" patterns with only a single defined possibility—where the "OneOf" serves no immediate purpose. With the "OneOf" pattern already present, you can modify the KnowledgeBase with the minimum of effort.

## Modifying a "Character Set" Type Choice-Set Set

A Character Set pattern has two possible roles:

- A Character Set can contain all characters of a given type;
- A Character Set can contain all characters that serve a given purpose.

**Example 8. The Character Set type choice-set**

▶ The Character Set *Alpha* contains all the characters of a given type—the 26 uppercase and the 26 lowercase letters. The name “Alpha” describes the **type** of the Character Set’s characters.

The Character Set *MenuOptionChar* contains all the characters that are possible inputs to a “menu” field. The name “MenuOptionChar” describes the **purpose** of the Character Set’s characters.

These two Character Sets, *Alpha* and *MenuOptionChar* are **identical**.

**The Advantage of Identical Character Sets**

The advantage of identical Character Set patterns is that you can modify one of the Character Sets without affecting the other Character Set.

**Example 9. Advantages of identical character sets**

▶ The Character Sets *Blank* and *ColumnIgnoreChar* are identical. *Blank* contains a single character, the blank produced by pressing the spacebar. The name “Blank” describes the contents of *Blank*, which in this case is a single character.

*Blank* is a child pattern to many other patterns. In most of these cases the parent patterns contain *Blank* because a blank is what separates different elements of a Complete Information Sequence. Correspondingly, *Blank* separates different elements of the parent pattern.

*ColumnIgnoreChar* is identical to *Blank*. The name “ColumnIgnoreChar” describes the specific purpose of this Character Set pattern—which is to identify a list column that does not need to be visually represented on the GUI.

The KnowledgeBase “as is” does not need the pattern *ColumnIgnoreChar*. The pattern *Blank* could replace *ColumnIgnoreChar* without affecting either the KnowledgeBase or ACE.

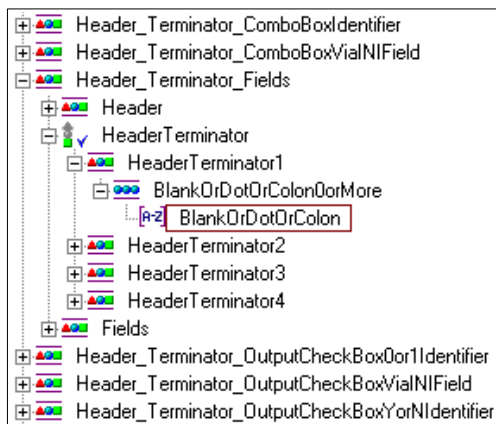
The advantage of having *ColumnIgnoreChar* is that you can modify this Character Set without affecting the patterns that contain *Blank*. If your host application uses some other character to visually separate different list columns, then you can add this character to *ColumnIgnoreChar* without affecting other places where the KnowledgeBase uses the “blank” character.

**Modifying a Character Set**

Example 10 uses pattern *BlankOrDotOrColon* to illustrate a solution for the modification of a character set.

**Example 10. BlankOrDotOrColon**

- ▶ The pattern *BlankOrDotOrColon* contains the “blank” character, the “period” and the “colon.” *BlankOrDotOrColon* is used by many primary patterns as the separator pattern between a child pattern that recognizes “header text” and a child pattern that recognizes a field:

**Figure 18. BlankOrDotOrColon**

Suppose your host application uses an additional character, the asterisk “\*”, as a separator between header text and fields. You should **not** add the “\*” character to *BlankOrDotOrColon* as the name will no longer reflect the contents, and other patterns that contain *BlankOrDotOrColon* may be adversely affected.

The preferred solution is:

- 1 Create a new Character Set pattern named *BlankOrDotOrColonOrAsterisk* which includes the “blank,” the period, the colon and the asterisk.
- 2 Replace *BlankOrDotOrColon* with *BlankOrDotOrColonOrAsterisk* in those patterns that need to recognize the asterisk as a separator character.

**Example 11. ComboBoxIdentifierChar****Figure 19. ComboBoxIdentifierChar character set**

The pattern *ComboBoxIdentifierChar* is used to recognize a character immediately preceding a line of prompts for the allowed inputs to a field.

Suppose your host application also uses the left angle bracket "<" as a ComboBox identifier. You can add the "<" character to *ComboBoxIdentifierChar* without worrying that other unrelated patterns will be affected because *ComboBoxIdentifierChar* is used for a single purpose.

## General Guidelines for Modifying Character Set Patterns

The guidelines are:

- A Character Set pattern whose name describes the **type** of characters it contains should **not** be modified. When such a Character Set does not handle a host screen situation, create a new Character Set and substitute it for the existing Character Set in the appropriate patterns.
- A Character Set pattern whose name describes the purpose all its characters serve **can** be modified.

You must ensure that *\*Separator* and *\*Terminator* Character Sets do not contain characters in common with the patterns they delimit.

## Modifying an “Iteration” Type Choice-Set

When an Iteration pattern does not recognize a repeated host screen character group, you can replace the Iteration pattern with a new Iteration pattern.

### Example 12. Modifying an Iteration



"F3 = exit"

The character group above does not satisfy *FKey03* because the Iteration pattern *Blank0to1* does not recognize two successive blanks.

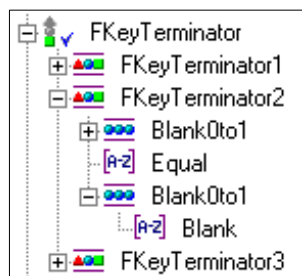
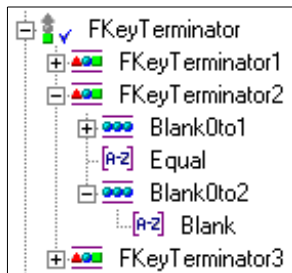


Figure 20. FKeyTerminator

A solution to this situation is to replace *Blank0to1* with *Blank0to2* which is satisfied by zero, one or two blanks.

The pattern definition tree for *FKeyTerminator* is now:

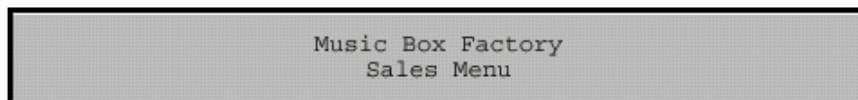


**Figure 21. Modified FKeyTerminator**

## Case

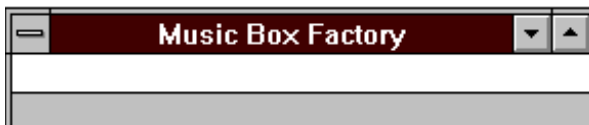
You can use the case of characters to narrow the scope of a Pattern Definition. When a Complete Information Sequence has a definite uppercase/lowercase structure, you can use case to improve the host screen analysis.

The pattern *WindowCaptionFullScreen* is designed to recognize the host screen's title:



**Figure 22. Title recognized by WindowCaptionFullScreen**

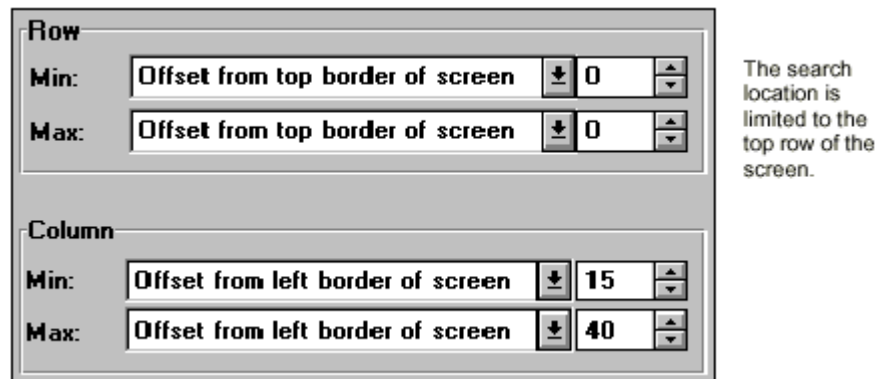
and make this title the GUI's window caption:



**Figure 23. GUI window created by WindowCaptionFullScreen**

As in the above example, the KnowledgeBase is set to search for the screen title in the top row of the screen:





**Row**

**Min:** Offset from top border of screen 0

**Max:** Offset from top border of screen 0

**Column**

**Min:** Offset from left border of screen 15

**Max:** Offset from left border of screen 40

The search location is limited to the top row of the screen.

Figure 24. Setting location limits

**Note:** The fields **Min** and **Max** of the location parameter are limits on where the Pattern Definition must begin. For example, the above parameters specify that **WindowCaptionFullScreen** must begin between columns 15 and 40, but it can extend beyond column 40.

The title may not always be in the top row of the host screen. Some of the host screens may have their titles in the second row:

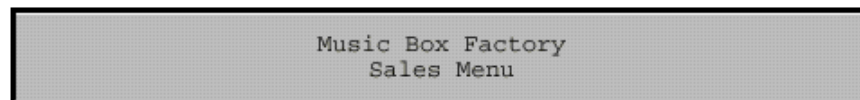


Figure 25. Host screen title on the second row

This situation cannot be handled by simply allowing for the screen title to be in the second row as well as the top row. Other titles, such as the title “Sales Menu” in the example above, may be in the second row in some screens.

A general solution to this situation is described in “New Pattern Definitions” on page 59.

A simple solution to this situation can be used if the screen title’s uppercase/lowercase structure always differs from that of other titles. Suppose the screen title is always in uppercase, so that some host screens have this form:

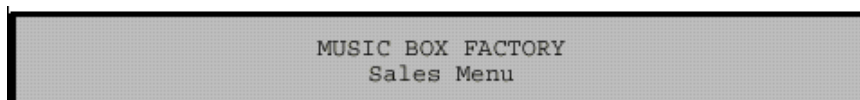


Figure 26. Screen title in uppercase

and some host screens have this form:

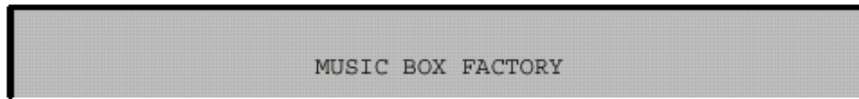


Figure 27. Screen title in uppercase and on second row

In this situation, the location parameter of *WindowCaptionFullScreen* can be extended to the second row:

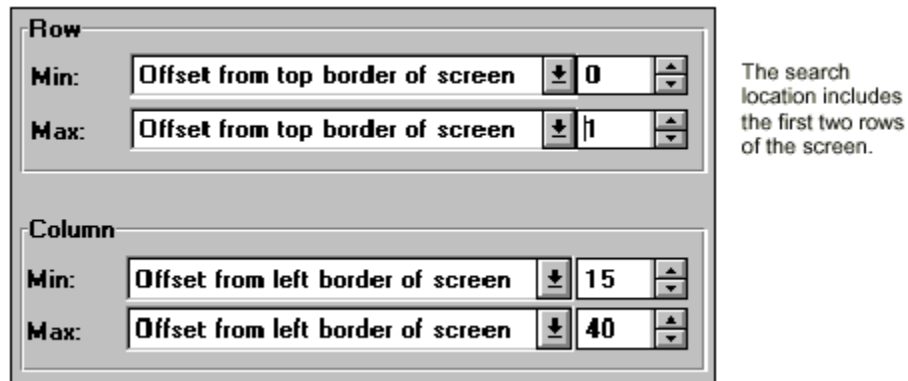


Figure 28. Extending location parameter of *WindowCaptionFullScreen*

as long as the child pattern *WindowCaptionChar*:

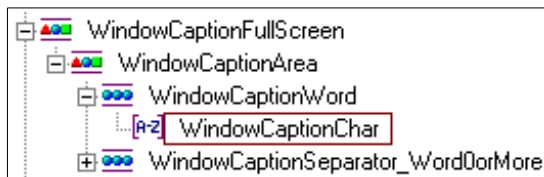


Figure 29. *WindowCaptionChar*

is modified to **exclude** lowercase letters:

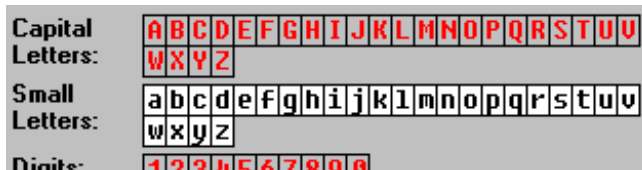


Figure 30. Modification of *WindowCaptionChar* character set

## New Pattern Definitions

You may need to create new Pattern Definitions when

- A Primary Pattern does not have the correct order or type of child patterns required to recognize an unusually structured Complete Information Sequence.
- A Primary Pattern must be included with other patterns in a new “umbrella” Primary Pattern because the original Primary Pattern cannot recognize an unusually structured Complete Information Sequence.

In both these cases you should not delete the original Primary Pattern. In the first situation the KnowledgeBase must still be able to recognize a Complete Information Sequence with the “usual” structure. In the second situation, the original Primary Pattern is a component of the new Primary Pattern.

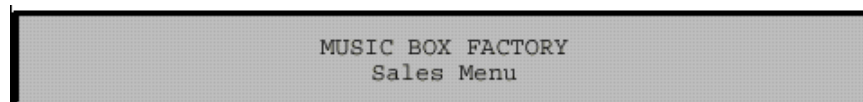
## Miscellaneous New Patterns

The KnowledgeBase is necessarily finite, so there will always be situations that can only be handled with new Pattern Definitions. The following example illustrates how a new pattern can handle a particular situation.

### Example 13. Creating a new Pattern Definition

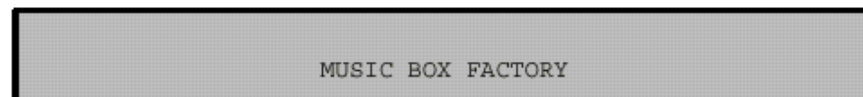


The KnowledgeBase is set to search for the screen title in the top row of the screen:



**Figure 31. Screen title on the first row**

This assumption may not be true for all of a host application’s screens. Some of the host screens may have their titles in the second row:



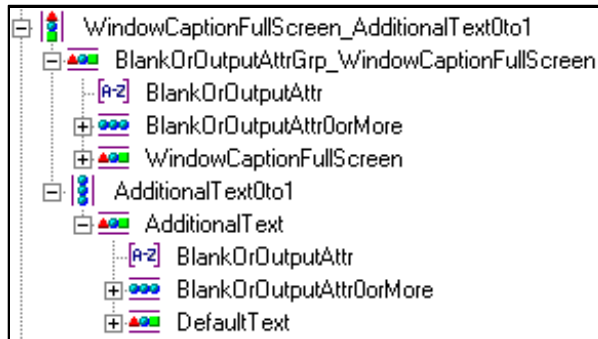
**Figure 32. Screen title on the second row**

This situation cannot be handled by simply allowing the screen title to be in the second row as well as the top row. Other titles, such as the title “Sales Menu” in the example above, may be in the second row in some screens.

This situation can be handled by grouping together the two Primary Patterns *WindowCaptionFullScreen* and *DefaultText* into a new “umbrella” Primary Pattern. Note that this violates the guideline that a Primary Pattern should correspond to a **single** Complete Information Sequence.

To group together *WindowCaptionFullScreen* and *DefaultText*:

- 1 Create a new Primary Pattern *WindowCaptionFullScreen\_AdditionalText0to1* with the following structure:



**Figure 33. WindowCaptionFullScreen\_AdditionalText0to1**


The main idea of Primary Pattern

*WindowCaptionFullScreen\_AdditionalText0to1* is that it contains *WindowCaptionFullScreen* followed, optionally, by *DefaultText*. This means that when a row of host screen text satisfies *WindowCaptionFullScreen* a row of text immediately following will be analyzed as *DefaultText*.

The two “Blank or Output Attribute” patterns are part of *WindowCaptionFullScreen\_AdditionalText0to1* because each row element of a Vertical Group pattern must start in the same column.

As in the example above, the textual part of the two rows might not, in fact, start in the same column. The “padding” with optional “Blank or Output Attributes” allows each row to start in the same column. ACE does not allow patterns to start with optional characters, so each *BlankOrOutputAttr0orMore* is preceded with *BlankOrOutputAttr*.

- 2 Set all the Location parameters for *WindowCaptionFullScreen* to “no limit.”
- 3 Set the Location parameters for *WindowCaptionFullScreen\_AdditionalText0to1* to:  
 Minimum Offset from top border of Screen 0.  
 Maximum Offset from top border of Screen 1.  
 Minimum Offset from left border of Screen 14.  
 Maximum Offset from left border of Screen 39



The search location starts between screen columns 14 and 39.

**Figure 34. WindowCaptionFullScreen\_AdditionalText0to1 location**

The “offset from left border” parameters for WindowCaptionFullScreen were originally 15 and 40. The “offset from left border” of WindowCaptionFullScreen\_AdditionalText0to1, 14 and 39, is shifted one column to the left. This shift is necessary because of the *BlankOrAttr* pattern which precedes WindowCaptionFullScreen in WindowCaptionFullScreen\_AdditionalText0to1:

- A *WindowCaptionFullScreen* that starts in column 15 is preceded by *BlankOrAttr* in column 14.
  - In order for *WindowCaptionFullScreen* to start no further right than column 40, *BlankOrAttr* can be no further right than column 39.
- 4 Substitute the new pattern WindowCaptionFullScreen\_AdditionalText0to1 for the pattern WindowCaptionFullScreen in Primary Section *FullScreenData*.

## Priority Concepts

ACE searches for patterns according to their order. Within a screen layout section, ACE searches for the patterns according to the order that they appear in the list box.

Figure 35 shows the Section Pattern Definitions list box of the ApplytoAll Primary Section Definition dialog box.



**Figure 35. Section Pattern Definitions list box**

ACE searches the *ApplyToAll* section first for the *DateViaSDFField* Pattern Definition, then for the *TimeViaSDFField* Pattern Definition, etc.

This feature allows ACE to search for more specific patterns before more general patterns.

#### Example 14. Order of FKey patterns in Section Pattern Definitions list box



The order of the patterns in the previous example means that ACE searches for the pattern *FKey03* before searching for the pattern *FKey*. ACE uses their relative order as follows:

The host function key *F3* is always an exit function, so Pattern Definition *FKey03* includes a fixed Representation Component:

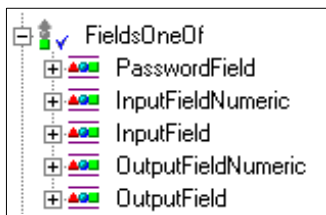


**Figure 36. Representation component of FKey03**

Complete Information Sequences such as “*F3=Exit*” satisfy *FKey03* but they also satisfy the pattern *FKey*. In order for the *F3* function key to be represented by the “*Exit*” button, ACE’s analysis must match “*F3*” host character groups to the Pattern *FKey03* and not to the more general Pattern *FKey*. Therefore, ACE must search for Pattern *FKey03* before Pattern *FKey*, and so *FKey03* appears before *FKey* in the *ApplyToAll* **Section Pattern Definitions** list.

Within a *OneOf* Pattern Definition, ACE searches for the topmost child pattern first and then each of the following child patterns in sequence.

#### Example 15. Order of Fields patterns in Section Pattern Definitions list box



**Figure 37. FieldsOneOf pattern**

ACE searches first for the *PasswordField* Pattern Definition, then for the *InputFieldNumeric* Pattern Definition, etc.

The GUI edit box for passwords does not display its input. In order that ACE allocate “password field” host character groups to the Pattern PasswordField and not to the other FieldsOneOfPatterns, PasswordField appears first in the list of FieldsOneOf child patterns.

**Note:** Whenever there is a question of priority, such as in “OneOf” type patterns, or within Primary Sections, priority is determined **only** by the order in which the patterns are listed. See Sections and Pattern Definition Search Order in *JIS Interface Server: Basic User’s Guide*.

## Using Priority to Modify the KnowledgeBase

You can use priority to add a Pattern that handles an unusual situation without disabling the Pattern that handles the usual situation.

In Figure 38:

```
F3=Exit F4=Prompt F1 =Initial Menu    F18=Printà  
000000000000000000000000000000000000
```

**Figure 38. Host screen pattern**

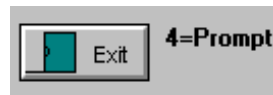
ACE recognizes the character group “F3=Exit F” as Pattern *FKey03* and the character group “4=Prompt” as static text.

The result of this analysis is that ACE presents the GUI as:

<u>F</u> ile	<u>E</u> dit	<u>C</u> ommands	<u>H</u> elp
		<u>E</u> xit F F3	

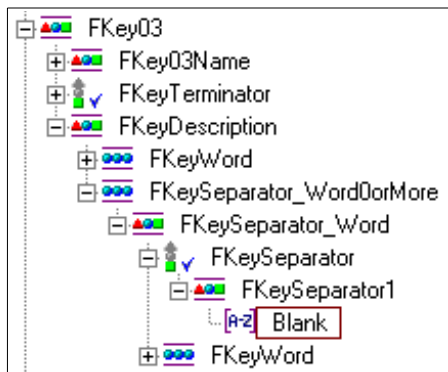
**Figure 39. Result of analysis of FKey03 - menu item**

and



**Figure 40. Result of analysis of FKey03 - button**

ACE analyzes the host screen this way because the end of the *FKey03* Pattern is *FKeySeparator Word0orMore* and *FKeySeparator* recognizes single spaces:

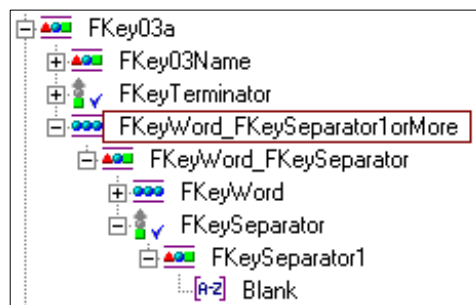


**Figure 41. FKeySeparator**

*FKey03* must include single spaces in this way in order to correctly recognize patterns such as “F3=Quit Screen”.

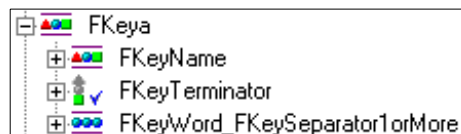
To remedy this situation:

- 1 Create a new Primary Pattern, *FKey03a*, that ends with a new pattern *FKeyWord\_FKeySeparator1orMore*:



**Figure 42. FKeyWord\_FKeySeparator1orMore**

- 2 Attach the Representation Components Button, MenuItem, Accelerator and GeneralUTMethod —each with the same parameters as they have in *FKey03*— to *FKey03a*.
- 3 Create a new Primary Pattern, *FKeya*, that ends with *FKeyWord\_FKeySeparator1orMore*:



**Figure 43. New pattern using FKeyWord\_FKeySeparator1orMore**

- 4 Add *FKey03a* to the *ApplyToAll* (Primary) Section Pattern Definitions list immediately before *FKey03*.
- 5 Add *FKeya* to the *ApplyToAll* (Primary) Section Pattern Definitions list immediately before *FKey*.



- 6 If you are using Screen Definition Files, such as DDS, add *FKey03a* and *FKeya* to the *Static Filter Section Pattern Definitions* list.

ACE will now correctly analyze the function key character groups.

**Note:** You must create an additional new Pattern *FKeya* so that ACE will not match “F4=Prompt F” with *FKey*. The original Patterns *FKey03* and *FKey* are still necessary because the new Patterns will not recognize character groups such as “F3=Exità” or “F18=Printà”

## Maintenance

The two patterns *FKey03* and *FKey03a* recognize the same Complete Information Sequences and therefore they should have identical Representation Components. Thus, whenever you modify *FKey03* you must remember to modify *FKey03a* in the identical way.

You can ensure that both patterns will always have the same Representation Components by grouping the two Pattern Definitions under a new parent pattern and attaching all the Representation Components to this new parent pattern.

To group together *FKey03* and *FKey03a*:

- 1 Record the *Screen*, *Manager*, *Window*, *Style* and *Format* settings for each of *FKey03*’s four Representation Components.
- 2 Remove the four Representation Components from *FKey03* and from *FKey03a*.
- 3 Create a new Primary Pattern *FKey03Both* with the following structure:



**Figure 44. FKey03Both**

*FKey03a* has priority over *FKey03* because *FKey03a* is the first child pattern of *FKey03OneOf*.

- 4 Attach the four Representation Components you removed from *FKey03* to *FKey03Both* and give them the settings you recorded in *Step 1*.
- 5 Note that in the *Screen* setting you have **two** branches to light. For example, in the *Screen* setting for the *Accelerator* Representation Component you must light *FKeyName* under both *FKey03* and *FKey03a*.
- 6 Substitute *FKey03Both* for *FKey03* and *FKey03a* in the *ApplyToAll* (Primary) **Section Pattern Definitions** list.
- 7 Substitute *FKey03Both* for *FKey03* and *FKey03a* in the **Static Filter Section Pattern Definitions** list.



## Chapter 5. Selected Pattern Definitions

This chapter is a guide to the specifics of modifying the KnowledgeBase. The approach taken is to present structural, functional and modification information about certain selected Pattern Definitions. While not all the Pattern Definitions are listed here, the Pattern Definitions that are listed are representative of the entire KnowledgeBase.

This chapter describes:

- An explanation of the syntax and conventions used to describe the selected Pattern Definitions.
- The individual Pattern Definitions.

### Syntax and Conventions

Each pattern's description has the following format:

#### The Pattern Name

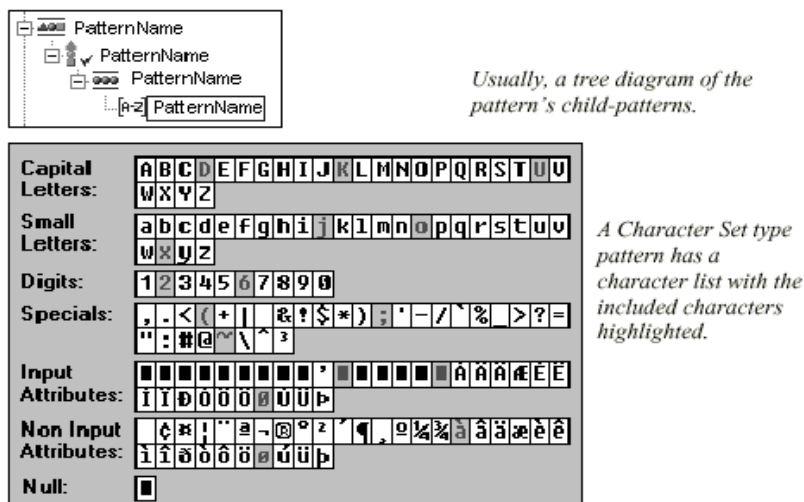


Figure 45. Example of the pattern name

<b>Structure</b>	<p>Optional additional details such as:</p> <ul style="list-style-type: none"><li>- A tree diagram of the parent patterns.</li><li>- The included characters of child Character Set patterns.</li></ul> <p>An outlined pattern on any tree diagram indicates that the outlined pattern is the link to the additional details.</p>
<b>Purpose</b>	<p>The host information that the pattern recognizes. Some patterns have special functions that are also described in the <i>Purpose</i> section.</p>
<b>Complete/Partial Information Sequence</b>	<p>Host characters containing the pattern with each occurrence of the pattern underscored.</p>
<b>Representation</b>	<p>The pattern's representation(s) or the names of the parent/child patterns that do have representations.</p>
<b>Modifications</b>	<p>Typical modifications you may need to perform, or the names of the parent/child patterns that should be modified. "None" means that it is strongly suggested that you do not modify the pattern. "None. In child patterns" means that you should modify the pattern indirectly by modifying the child patterns. "Forbidden" means that any modification of the pattern will prevent ACE from functioning correctly.</p>

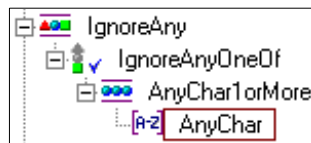
## Selected Pattern Definitions

The following are pattern definitions that were selected because they are a good representation of the types of pattern definitions that reside in the default ACE KnowledgeBase.

## AnyChar

Capital Letters:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Small Letters:	a b c d e f g h i j k l m n o p q r s t u v w x y z
Digits:	1 2 3 4 5 6 7 8 9 0
Specials:	, . < ( +   _ & ! \$ % ' ; ' - / \ ^ % _ > ? = : : # @ ~ \ ^ % _
Input Attributes:	À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ñ Ò Ó Ô Õ Ö × Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ñ ò ó ô õ ö ÷ ù ú û ü ý þ ß à á â ã ä å æ ç è é ê ë ì í î ï ñ ò ó ô õ ö ÷ ù ú û ü ý þ ß
Non Input Attributes:	À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ñ Ò Ó Ô Õ Ö × Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ñ ò ó ô õ ö ÷ ù ú û ü ý þ ß
Null:	

**Figure 46. AnyChar character set**



### Figure 47. AnyChar

## Structure

All available characters. *AnyChar*'s parent pattern, *IgnoreAny*, is the sole pattern in Primary Section *IgnoreAny*.

<b>Purpose</b>	<p>The pattern <i>AnyChar</i> recognizes any character. You make use of <i>AnyChar</i> whenever you make ACE ignore characters through manually placing an <i>IgnoreAny</i> Section on them. <i>AnyChar</i> is the pattern that makes the <i>IgnoreAny</i> Section perform properly.</p> <p>The pattern <i>AnyChar</i> recognizes any character. You make use of <i>AnyChar</i> whenever you make ACE ignore characters through manually placing an <i>IgnoreAny</i> Section on them. <i>AnyChar</i> is the pattern that makes the <i>IgnoreAny</i> Section perform properly.</p>
<b>Partial Information Sequence</b>	Not applicable. Any characters will be recognized.
<b>Representation</b>	None.
<b>Modifications</b>	Forbidden.

## CheckBox0or1Identifier

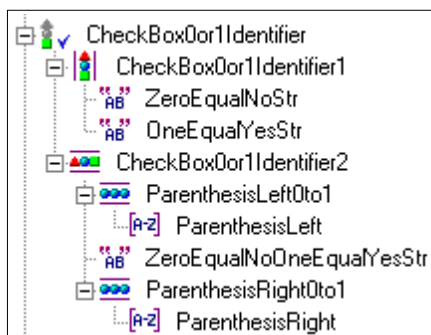


Figure 48. `CheckBox0or1Identifier`

<b>Purpose</b>	The company
<b>Partial Information Sequence</b>	<p>The pattern <i>CheckBox0or1Identifier</i> is used to recognize the prompt limiting the allowed input characters of a field to "0" and "1".</p> <p>àFull Details: 'Bààà(0=No, 1=Yes)</p>

<b>Representation</b>	None
<b>Modifications</b>	New child patterns recognizing variations of the prompt.

## CheckBoxSeparator

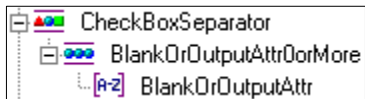


Figure 49. `CheckBoxSeparator`

<b>Purpose</b>	The Pattern <i>CheckBoxSeparator</i> is used to recognize the boundary between an input area and the description of the allowed inputs.
<b>Partial Information Sequence</b>	<code>àFull Details: 'Bààà (Y/N)</code>
<b>Representation</b>	CheckBox.
<b>Modifications</b>	Other iterations of special characters, introduced via new child patterns.

## CheckBoxViaINIField

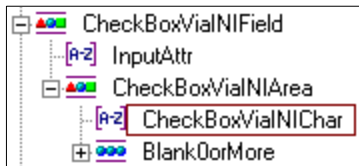
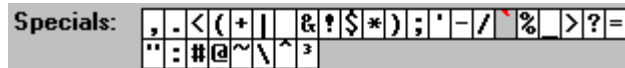


Figure 50. `CheckBoxViaINIField`

**Structure**

The Character Set *CheckBoxViaINIChar* contains a single character:

**Purpose**

The Pattern *CheckBoxViaINIField* is used in conjunction with customized screen maps and the CHECKBOX.INI file to identify an input checkbox by means of a particular screen character. You use this mechanism when the host screen does not contain Yes/No or other characters that identify an input field as accepting only “yes or no.” Compare with *CheckBoxYorNIIdentifier*.

**Partial  
Information  
Sequence**

àFull Details: ‘^

**Representation**

CheckBox formatted to put “Y” or “N” in the host field.

**Modifications**

Change the identifier character in *CheckBoxViaINIChar*. Your application may require different inputs in the host field, such as either “0” or “1”. In this case you can change the formatting of the Representation definition, or create an entirely new Pattern Definition *CheckBox1or0ViaINIField*.



# CheckBoxYorNIdentifier

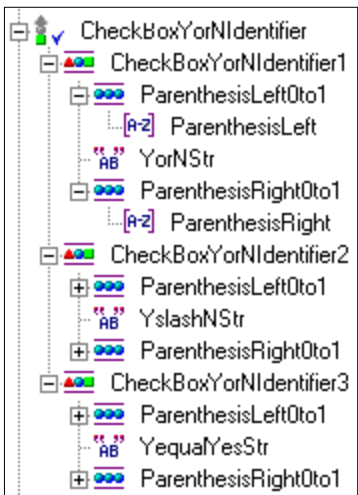


Figure 51. CheckBoxYorNIdentifier

<b>Purpose</b>	The pattern <i>CheckBoxYorNIdentifier</i> is used to recognize the prompt limiting the allowed input characters of a field to “Y” and “N”.
<b>Partial Information Sequence</b>	àFull Details: ‘Bààà(Y=Yes)
<b>Representation</b>	None
<b>Modifications</b>	New child patterns recognizing variations of the prompt.

# ColumnCheckBoxViaINI

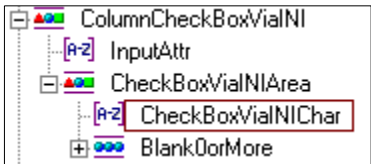
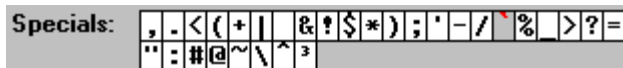


Figure 52. ColumnCheckBoxViaINI

**Structure**

The Character Set *CheckBoxViaINIChar* contains a single character:

**Purpose**

The pattern *ColumnCheckBoxViaINI* recognizes a column as a child pattern within a larger *List\** pattern. It is used in conjunction with customized screen maps and the CHECKBOX.INI file to identify an input checkbox by means of a particular screen character. You use this mechanism when the host screen does not contain Yes/No or other characters that identify an input field as accepting only “yes or no.”

See *List\_0Separator\_1Header* for a discussion of the “column” aspect of patterns.

**Partial Information Sequence**

àBBBB' ^  
àBBBB' ^  
àBBBB' ^

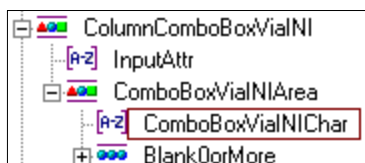
**Representation**

A column of CheckBoxes.

**Modifications**

None. You can change the identifier character in child pattern *CheckBoxViaINIChar*.

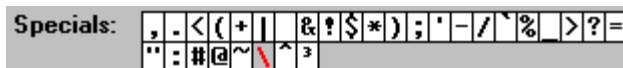
## ColumnComboBoxViaINI



**Figure 53. ColumnComboBoxViaINI**

**Structure**

The Character Set *ComboBoxViaINIChar* contains a single character:



<b>Purpose</b>	<p>The pattern <i>ColumnComboBoxViaINI</i> recognizes a column by its context within a larger <i>List*</i> pattern. It is used in conjunction with customized screen maps and the COMBOBOX.INI file to identify an input combobox by means of a particular screen character. You use this mechanism when an input field has a limited number of allowed values, and the host screen does not contain a list of these allowed values.</p> <p>See <i>List_0Separator_1Header</i> for a discussion of the “column” aspect of patterns.</p>
<b>Partial Information Sequence</b>	<pre>'\àà000000à 000000à '\àà000000à 000000à '\àà000000à 000000à</pre>
<b>Representation</b>	A column of ComboBoxes.
<b>Modifications</b>	None. You can change the identifier character in child pattern <i>CheckBoxViaINIChar</i> .

## ColumnHeaderTable

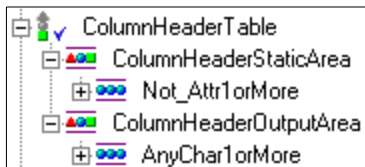


Figure 54. ColumnHeaderTable

<b>Purpose</b>	<p>The pattern <i>ColumnHeaderTable</i> refines the division of a list’s header row into “columns.” <i>ColumnHeaderTable</i> extracts the actual text to be displayed on the table buttons of the GUI from each header area defined by the hard and soft delimiters.</p>
<b>Partial Information Sequence</b>	<pre>àNameààSizeà 0000à àÀBBBBàà0000à 0000à ÀBBBBàà0000à 0000à ÀBBBBàà0000à 0000à</pre>

<b>Representation</b>	None. The child patterns are represented as table buttons.
<b>Modifications</b>	None.

## ColumnIgnore

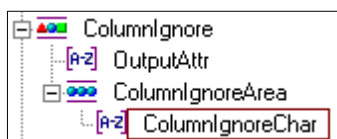
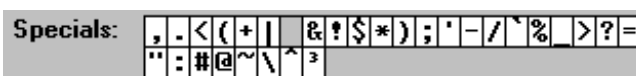


Figure 55. ColumnIgnore

<b>Structure</b>	The pattern <i>ColumnIgnoreChar</i> contains the character:
------------------	---

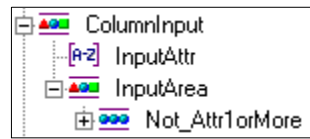


<b>Purpose</b>	The pattern <i>ColumnIgnore</i> recognizes a column by its context as a child pattern of a larger <i>List*</i> pattern. Host lists sometimes include a column of blanks as a way of visually separating the list's "content displaying" columns <i>ColumnIgnore</i> is used to identify and ignore such a "separating" column.
----------------	--

<b>Partial Information Sequence</b>	àNameààSizeàà _____ à 0000à àÀBBBBàà0000àà _____ à 0000à ÀBBBBàà0000àà _____ à 0000à ÀBBBBàà0000àà _____ à 0000à
-------------------------------------	---

<b>Representation</b>	Variable.
<b>Modifications</b>	None. You can add additional characters to <i>ColumnIgnoreChar</i> . Hosts often use " " and ":" as separator characters.

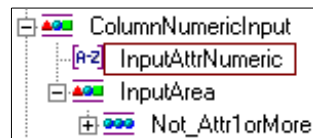
## ColumnInput



### Figure 56. ColumnInput

<b>Purpose</b>	The pattern <i>ColumnInput</i> recognizes a column by its context as a child pattern of a larger <i>List*</i> pattern. It is used to identify a column of input fields.
<b>Partial Information Sequence</b>	à0000'BBBà à0000'BBBà à0000'BBBà
<b>Representation</b>	AdjustableEdit.
<b>Modifications</b>	None.


## ColumnNumericInput



### Figure 57. ColumnNumericInput

**Structure**

The pattern *InputAttrNumeric* contains the characters:



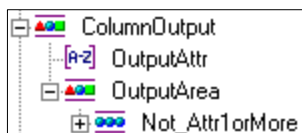
Input Attributes:	I	Y	O	O	O	O	U	P										A	A	A	E	E
-------------------	---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--	---	---	---	---	---

**Purpose**

The pattern *ColumnNumericInput* recognizes a column by its context within a larger *List\** pattern. It is used to identify a column of input fields whose allowed input values must be numeric.

<b>Partial Information Sequence</b>	<u>à0000</u> <u>EBB</u> <u>à0000</u> <u>EBB</u> <u>à0000</u> <u>EBB</u>
<b>Representation</b>	AdjustableEdit with Ext_FloatOrInt validity check.
<b>Modifications</b>	None.

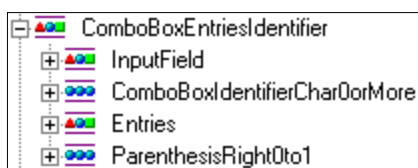
## ColumnOutput



**Figure 58. ColumnOutput**

<b>Purpose</b>	The pattern <i>ColumnOutput</i> recognizes a column by its context as a child pattern of a larger <i>List*</i> pattern. It is used to identify a column of output fields.
<b>Partial Information Sequence</b>	<u>à0000</u> <u>àB</u> <u>à0000</u> <u>àB</u> <u>à0000</u> <u>àB</u>
<b>Representation</b>	Static.
<b>Modifications</b>	None.

## ComboBoxEntriesIdentifier



**Figure 59. ComboBoxEntriesIdentifier**

<b>Purpose</b>	The pattern <i>ComboBoxEntriesIdentifier</i> is used to recognize an input field and the prompt for all the allowed values of the field.
<b>Partial Information Sequence</b>	% Discount:à <u>'BBàà(10,20,30)à</u>
<b>Representation</b>	ComboBox.
<b>Modifications</b>	None. See child patterns.

## ComboBoxIdentifier

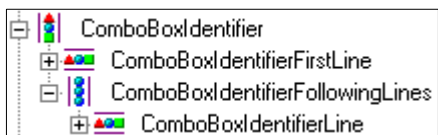


Figure 60. ComboBoxIdentifier

<b>Purpose</b>	The pattern <i>ComboBoxIdentifier</i> is used to recognize an input field, and <b>all</b> the prompts for allowed values for the field, even if the prompts are spread over more than one line on the host.
<b>Partial Information Sequence</b>	àFlavor:'B(1=Vanilla, 2=Chocolate) (3=Banana, 4=Mocha)
<b>Representation</b>	ComboBox.
<b>Modifications</b>	None. See child patterns.

## ComboBoxIdentifierCharacter

<b>Specials:</b>	, . < ( +   & ! \$ % * ) ; ' - / \ % _ > ? =
	" : # @ ~ ^ ` ¢
<b>Input Attributes:</b>	I İ Ö Ø Õ õ Ü ü Þ Á Â Ã Ä Å Æ É Ê Ë
<b>Non Input Attributes:</b>	ı ĩ î ï ö ø õ ŭ ü þ ç è é ê ë ã ä å æ ç è é

**Figure 61. ComboBoxIdentifierCharacter character set**

<b>Purpose</b>	The pattern <i>ComboBoxIdentifierCharacter</i> is used to recognize a character immediately preceding a line of prompts for the allowed inputs to a field.
<b>Partial Information Sequence</b>	àOrder By: 'Bàà(1=Number, 2=Last Name)à
<b>Representation</b>	None
<b>Modifications</b>	Additional characters.

## ComboBoxIdentifierChar0orMore

ComboBoxIdentifierCharOrMore  
[A-Z] ComboBoxIdentifierChar

### Figure 62. ComboBoxIdentifierChar0orMore

<b>Purpose</b>	The rows of a vertical group must all start in the same column. The pattern <i>ComboBoxIdentifierCharacter0orMore</i> is used to handle the possibility that the different rows of prompts for the allowed inputs to a field start in different columns.
<b>Partial Information Sequence</b>	<pre>àFlavor:'B(1=Vanilla, 2=Chocolate)     __ (3=Banana, 4=Mocha)</pre>
<b>Representation</b>	None



<b>Modifications</b>	None. You may want to limit the maximum number of iterations of <i>ComboBoxIdentifierChar</i> in one of <i>ComboBoxIdentifierChar0orMore</i> 's parent patterns. To do this, create a new pattern <i>ComboBoxIdentifierChar0toN</i> , that has a maximum number of iterations "N", and substitute it for <i>ComboBoxIdentifierChar0orMore</i> in the parent pattern.
----------------------	--

## ComboBoxIdentifierFirstLine

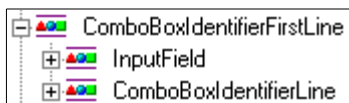


Figure 63. ComboBoxIdentifierFirstLine

<b>Purpose</b>	The rows of a vertical group must all start in the same column. The pattern <i>ComboBoxIdentifierFirstLine</i> is used to handle the possibility that the second row of <i>EntryPairs</i> prompts for the allowed inputs to a field starts in a column to the left of the first row of prompts. The offset is often due to the fact that only the first row of prompts is preceded by an input field. The only way to vertically group the two rows is for the first row to include the input field.
<b>Partial Information Sequence</b>	<code>àFlavor:'B(1=Vanilla, 2=Chocolate)</code> <code>(3=Banana, 4=Mocha)</code>
<b>Representation</b>	None. See the child patterns.
<b>Modifications</b>	None. See the child patterns.

## ComboBoxIdentifierLine

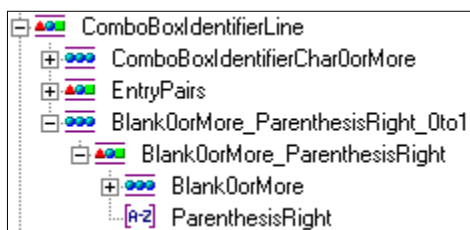


Figure 64. ComboBoxIdentifierLine

### Purpose

The pattern *ComboBoxIdentifierLine* is used to group together *EntryPairs* and other characters around it, so that a **vertical** iteration of, essentially, horizontal groups of *EntryPairs* can be taken together with a field into *ComboBoxIdentifier*. Without this grouping, a vertical iteration of *EntryPairs* could be recognized only if in each line on the host the *EntryPairs* were directly under one another. Further, the vertically iterated *EntryPairs* could be taken together with the field only if there were no characters between the field and the vertical iteration. See *ComboBoxIdentifier*.

### Partial Information Sequence

àOrder By: 'Bàà (1=Number, 2=Last Name)à

### Representation

None. See the child patterns.

### Modifications

None. See the child patterns.

## ComboBoxViaINIField

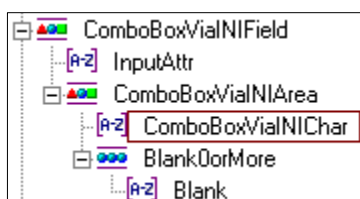
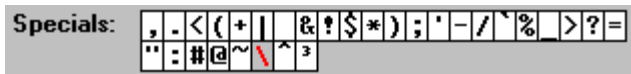


Figure 65. ComboBoxViaINIField

**Structure** The Character Set *ComboBoxViaINIChar* contains a single character:



**Purpose** The Pattern *ComboBoxViaINIField* is used in conjunction with customized screen maps and the COMBOBOX.INI file to identify an input combobox by means of a particular screen character. You use this mechanism when an input field has a limited number of allowed values, but the host screen does not contain a list of these allowed values.

**Partial Information Sequence** àStatus:à ' \

**Representation** ComboBox.

**Modifications** None. Change the identifier character in *ComboBoxViaINIChar*.

## ComboBoxWithIdentifierScattered

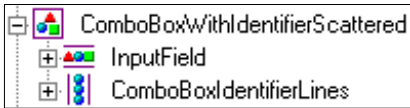


Figure 66. *ComboBoxWithIdentifierScattered*

**Purpose** The pattern *ComboBoxWithIdentifierScattered* is used to recognize an input field and the prompt for all the allowed values of the field, even if there are extraneous characters between the input field and the prompts. In addition there does not have to be a fixed offset between the input field and the lines of prompts. You can handle the extraneous characters independently of the characters making up the scattered group. Often you will want to ignore them via an *IgnoreAny* section.

<b>Complete Information Sequence</b>	àStatus:à <u>'B</u>
	Allowed Statuses
	1=Single
	2=Married
<b>Representation</b>	3=Divorced
	ComboBox.
<b>Modifications</b>	None. In child patterns.

## DefaultText

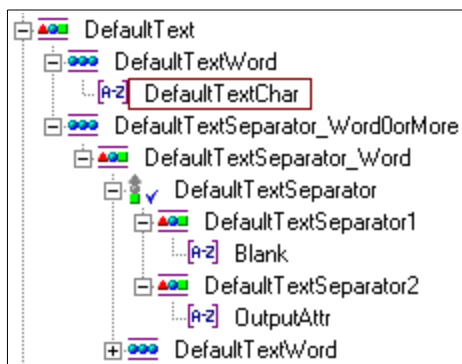


Figure 67. DefaultText

<b>Structure</b>	The Character Set <i>DefaultTextChar</i> contains the characters:
------------------	---

<b>Capital Letters:</b>	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
<b>Small Letters:</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>Digits:</b>	1 2 3 4 5 6 7 8 9 0
<b>Specials:</b>	. - < ( +   & ? \$ * ) ; ' - / \ % _ > ? = : # @ ~ \ ^ _
<b>Input Attributes:</b>	À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ñ Ò Ó Ô Õ Ö Ù Ú Û Ü
<b>Non Input Attributes:</b>	à á â ã ä å æ ç è é ê ë ì í î ï ñ ò ó ô õ ö ù ú û ü
<b>Null:</b>	

<b>Purpose</b>	The pattern <i>DefaultText</i> is the lowest priority pattern in the <i>ApplyToAll</i> section. <i>DefaultText</i> is designed to ensure that all text on the screen will be recognized by a KnowledgeBase pattern.
<b>Complete Information Sequence</b>	Characters
<b>Representation</b>	Static
<b>Modifications</b>	None. Characters that do not need a representation are best handled by using the <i>IgnoreAny</i> or <i>IgnoreText</i> patterns.

## Digit\_0\_0to1

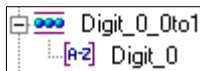


Figure 68. Digit\_0\_0to1

<b>Purpose</b>	The pattern <i>Digit_0_0to1</i> is used to allow function key patterns to recognize the digits from 0 to 9 with or without a leading "0".
<b>Partial Information Sequence</b>	PF03=Exit
<b>Representation</b>	None.
<b>Modifications</b>	None.

## DynamicFKeyGroup

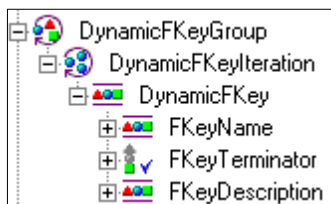


Figure 69. DynamicFKeyGroup

<b>Purpose</b>	The pattern <i>DynamicFKeyGroup</i> is similar to <i>FKey</i> and is used to recognize function key Complete Information Sequences when the number and position of such sequences can vary in Runtime.
<b>Complete Information Sequence</b>	F3=Exit
<b>Representation</b>	Dynamic.
<b>Modifications</b>	You can add additional dynamic iteration patterns, such as <i>DynamicFKey03Iteration</i> to <i>DynamicFKeyGroup</i> .

## DynamicListCommandIteration



Figure 70. DynamicListCommandIteration

<b>Purpose</b>	The pattern <i>DynamicListCommandIteration</i> is used to recognize multiple occurrences of sequences of characters satisfying <i>ListCommandArea</i> when the number and position of such sequences can vary during Runtime.
----------------	---

<b>Partial Information Sequence</b>	1-Update 2-Update 3-View
<b>Representation</b>	DynamicListCommand.
<b>Modifications</b>	You can add additional dynamic iteration patterns, such as <i>DynamicFKey03Iteration</i> to <i>DynamicFKeyGroup</i> .

## EndOfList

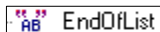


Figure 71. EndOfList

<b>Structure</b>	Contains a dummy string. See below.
<b>Purpose</b>	<p>When ACE analyzes a host screen and associates a host list with a GUI table, it fixes the exact number of rows the table displays. During Runtime, there may be fewer records in the list than table rows on the GUI. The result is that the GUI table contains empty white space below the last record. The <i>EndOfList</i> pattern can sometimes be used to avoid this.</p> <p>If you know that the last line of a list is <b>always</b> followed by a certain character sequence, such as "*****End of File*****", then you can set <i>EndOfList</i> to this string. When ACE detects this string during Runtime, it "knows" that the number of list records is exactly the number of rows between the start of the list and the <i>EndOfList</i> string. If this number of list records is less than the number of table rows on the GUI, ACE reduces the number of table rows, avoiding the white space. ACE adjusts the size of the table during Runtime as the number of list records changes.</p>
<b>Partial Information Sequence</b>	This pattern will always fail
<b>Representation</b>	None.

<b>Modifications</b>	If there is an end-of-list character sequence, then substitute this sequence for the default string.
----------------------	--

## Entries

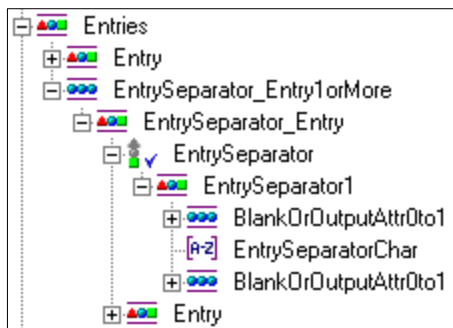


Figure 72. Entries

<b>Purpose</b>	The pattern <i>Entries</i> is used to recognize the prompt for all the allowed values of a field.
<b>Partial Information Sequence</b>	% Discount:à 'BBàà( <u>10,20,30</u> )à
<b>Representation</b>	None. See child patterns.
<b>Modifications</b>	None. See <i>Entry</i>

## Entry

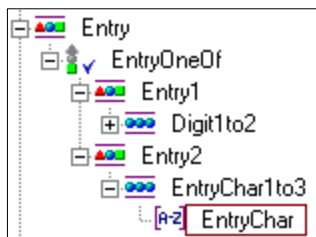


Figure 73. Entry



Structure

The Character Set *EntryChar* contains the characters:

Capital Letters:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Small Letters:	a b c d e f g h i j k l m n o p q r s t u v w x y z
Digits:	1 2 3 4 5 6 7 8 9 0
Specials:	, . < ( +   & ! \$ % * ) ; ' - / ^ _ ? = " : # @ ~ \ ' 3

Purpose

The pattern *Entry* is used to recognize a prompt for an allowed value of a field.

Partial Information Sequence

% Discount:à 'BBàà(10,20,30)à

Representation

HostBasedFormatValues, not shown on the GUI.

Modifications

None. If the allowed values for a field are other than those found in *Entry1* and *Entry2*, then *EntryChar* can be modified. *EntryChar* cannot have characters in common with *EntrySeparator* or *ParenthesisRight*. The number of iterations of *EntryChar* allowed in *Entry2* can be changed.

EntryPair

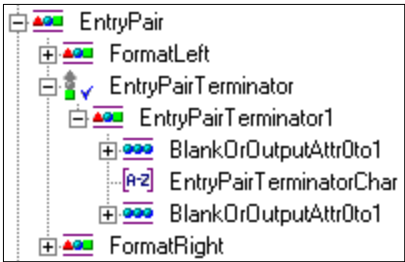
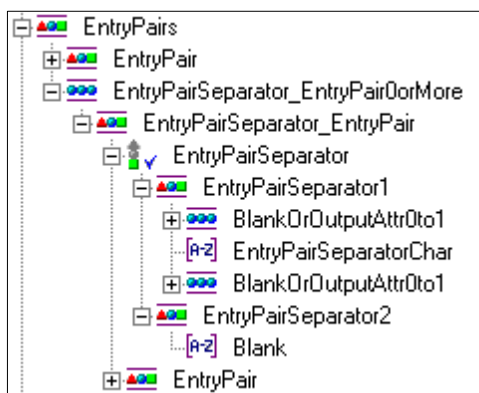


Figure 74. EntryPair

<b>Purpose</b>	The pattern <i>EntryPair</i> is used to recognize the complete prompt—input characters plus description—for one of the allowed inputs to a field.
<b>Partial Information Sequence</b>	<code>àOrder By: 'Bàà (1=Number, 2=Last Name)à</code>
<b>Representation</b>	HostBasedFormatItem. See the child patterns.
<b>Modifications</b>	None. See the child patterns.

## EntryPairs



**Figure 75. EntryPairs**

<b>Purpose</b>	The pattern <i>EntryPairs</i> is used to recognize a line of complete prompts—input characters plus description—for the allowed inputs to a field.
<b>Partial Information Sequence</b>	<code>àOrder By: 'Bàà (1=Number, 2=Last Name)à</code>
<b>Representation</b>	None. See the child patterns.
<b>Modifications</b>	None. See the child patterns.

# FKey

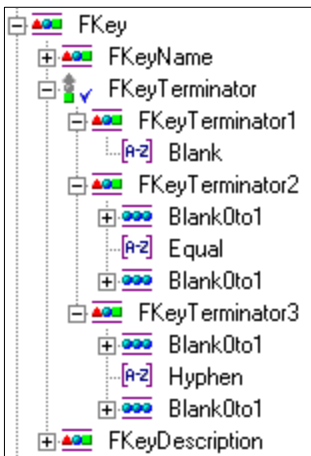



Figure 76. FKey

<b>Purpose</b>	The pattern <i>FKey</i> is used to recognize Complete Information Sequences that identify and describe the function of command keys.
<b>Complete Information Sequence</b>	Pf03=exit
<b>Representation</b>	MenuItem, Accelerator and GeneralUTMethod.
<b>Modifications</b>	None. See child patterns

# FKey01



Figure 77. FKey01

<b>Purpose</b>	The pattern <i>FKey01</i> recognizes Complete Information Sequences describing the function of the first command key. It is used to allow the first command key to be represented by a predefined Button, in place of a PushButton that takes its text from <i>FKeyDescription</i> .
<b>Complete Information Sequence</b>	Pf1 = Previous Menu
<b>Representation</b>	PictureButton:  Menu Item; Accelerator; GeneralUTMethod.
<b>Modifications</b>	<i>FKey01</i> should be used “as is” only when the first command key’s function is to cancel out of the current host screen without saving any changes. If the first command key has a different <b>fixed</b> function, then you can attach a different picture button to <i>FKey01</i> . Any of the <i>FKey##</i> patterns can be used whenever the corresponding command key has a fixed function.

## FKeyDescription

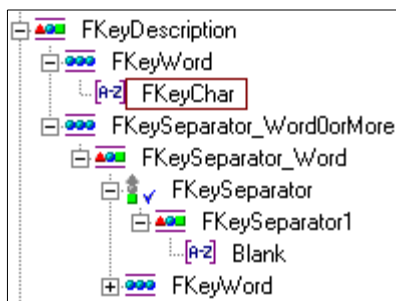


Figure 78. FKeyDescription

**Structure** The Character Set *FKeyChar* contains the characters:

Capital Letters:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Small Letters:	a b c d e f g h i j k l m n o p q r s t u v w x y z
Digits:	1 2 3 4 5 6 7 8 9 0
Specials:	, . < ( +   & ! \$ * ) ; ' - / \ % _ ? = " : # @ ~ \ ^ `
Input Attributes:	À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ñ Ò Ó Ô Õ Ö Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ñ ò ó ô õ ö ù ú û ü ý þ ß
Non Input Attributes:	
Null:	

**Purpose** The pattern *FKeyDescription* is used to recognize the description of a command key's function.

**Partial Information Sequence** F3=Exit

**Representation** None. See *FKey*.

**Modifications** None.

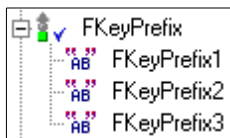
FKeyName



Figure 79. FKeyName

<b>Purpose</b>	The purpose of the pattern <i>FKeyName</i> is representational. <i>FKey</i> 's Accelerator representation needs the information contained in both <i>FKeyPrefix</i> and <i>Digit1to2</i> . In order for the Accelerator representation to be able to use both pieces of information, there must be a pattern that includes both <i>FKeyPrefix</i> and <i>Digit1to2</i> as child patterns— <i>FKeyName</i> .
<b>Partial Information Sequence</b>	Cmd03 = Quit
<b>Representation</b>	None
<b>Modifications</b>	None. See <i>FKeyPrefix</i> .

## FKeyPrefix



**Figure 80. FKeyPrefix**

<b>Purpose</b>	The pattern <i>FKeyPrefix</i> is used to recognize the character(s) that identify a larger sequence of characters as describing a command key's function.
<b>Partial Information Sequence</b>	<u>Cmd</u> 03 = Quit
<b>Representation</b>	None.
<b>Modifications</b>	New sequences of identifying characters can be added as additional child patterns. You can also remove the sequences that are not present in your application.

# FormatLeft

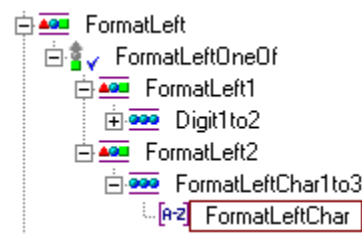


Figure 81. FormatLeft

## Structure

The Character Set *FormatLeftChar* contains the characters:

Capital Letters:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Small Letters:	a b c d e f g h i j k l m n o p q r s t u v w x y z
Digits:	1 2 3 4 5 6 7 8 9 0
Specials:	, . < ( +   & ! \$ * ) ; ' - / % _ ? = " : # @ ~ \ ^ `

## Purpose

The pattern *FormatLeft* is used to recognize the input segment of a prompt for the allowed inputs of a field.

## Partial Information Sequence

àOrder By: 'Bàà(1=Number, 2=Last Name)à

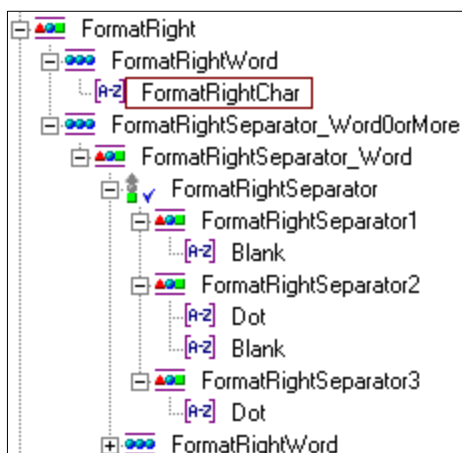
## Representation

HostBasedFormatValues (Screen Value); not shown on the GUI.

## Modifications

None. *FormatLeftCharacter* can be modified, but it must not have any characters in common with *EntryPairTerminator* (see *EntryPair*) nor have any characters in common with *EntryPairSeparator* (see *EntryPairs*). In general, delimiter patterns and the patterns being delimited cannot have characters in common.

## FormatRight



**Figure 82. FormatRight**

### Structure

The Character Set *FormatRightChar* contains the characters:

<b>Capital Letters:</b>	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
<b>Small Letters:</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>Digits:</b>	1 2 3 4 5 6 7 8 9 0
<b>Specials:</b>	, . < ( +   & ! \$ * ) ; ' - / ^ % _ ? = " : # @ ~ \ ^ 3

### Purpose

The pattern *FormatRight* is used to recognize the descriptive segments of a prompt for the allowed character group inputs of a field.

### Partial Information Sequence

àOrder By: 'Bàà (1=Number, 2=Last Name)à

### Representation

HostBasedFormatValues (Window Value). The descriptive segments are the items in a ComboBox.

### Modifications

None. *FormatRightChar* and *FormatRightSeparator* can be modified but they must not have any characters in common.



## HeaderNoRD

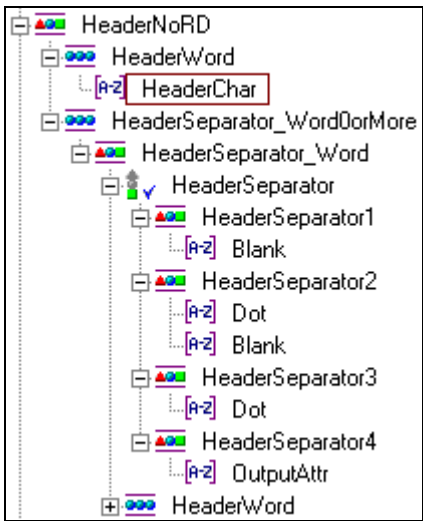


Figure 83. HeaderNoRD

<b>Purpose</b>	The pattern <i>IgnoreText</i> is used to recognize host screen text that should not be represented on the GUI. The supplied KnowledgeBase contains an example, <i>IgnoreText</i> , which is the Complete Information Sequence below.
<b>Complete Information Sequence</b>	Type options, Press Enter
<b>Representation</b>	Forbidden.
<b>Modifications</b>	None. Additional ignored texts are added as child patterns of <i>IgnoreTextOneOf</i> .

## ListCertainHeader

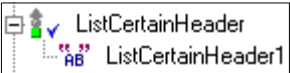


Figure 84. ListCertainHeader

<b>Purpose</b>	The pattern <i>ListCertainHeader</i> is used to recognize and define strings that are always complete headers for a single column. Use this pattern to avoid problems associated with headers that are not directly above their columns and problems with header rows where the separation between individual column headers is ambiguous.
<b>Partial Information Sequence</b>	Change to certain header
<b>Representation</b>	No direct representation. Header (and other) representations are attached to other parts of <i>List*</i> Pattern definitions.
<b>Modifications</b>	Change the default string to an actual header string, and add additional header strings.

## ListCommand

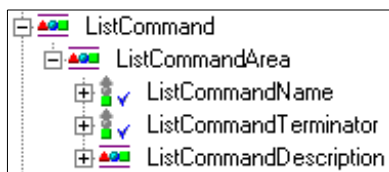


Figure 85. ListCommand

<b>Purpose</b>	The pattern <i>ListCommand</i> is used to recognize the prompt for input to a list row's input field, together with the meaning of the allowed input.
<b>Partial Information Sequence</b>	<u>S-Save</u> <u>D-Delete</u>
<b>Representation</b>	MenuItem. GeneralUTMethod.
<b>Modifications</b>	None. In child patterns.

# ListCommandDescription

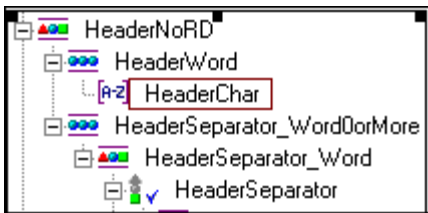


Figure 86. ListCommandDescription

**Structure** The pattern *ListCommandChar* contains the characters:

Capital Letters:	A B C D E F G H I J K L M N O P Q R S T U V
Small Letters:	W X Y Z
Digits:	1 2 3 4 5 6 7 8 9 0
Specials:	. < ( +   & ! \$ * ) ; ' - / ^ % _ > ? =
Input Attributes:	I I 0 0 0 0 0 U U b
Non Input Attributes:	i i 0 0 0 0 0 u u b
Null:	

**Purpose** Recognizes the description of the meaning of allowed field values.

**Partial Information Sequence** 1-Update 2-Delete 3-View

**Representation** Forbidden.

**Modifications** None. Child pattern *ListCommandChar* cannot contain any characters in common with *ListCommandTerminator* or with *ListCommandSeparator*.

## ListCommandName

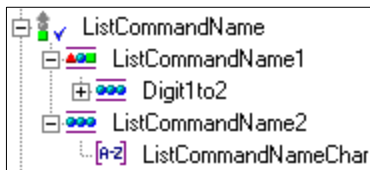


Figure 87. ListCommandName

<b>Purpose</b>	The pattern <i>ListCommandName</i> is used to recognize the sequences that are the possible input values to a list's input fields. The sequences are usually no more than two characters long.
<b>Partial Information Sequence</b>	<u>S</u> -Save <u>D</u> -Delete
<b>Representation</b>	Forbidden.
<b>Modifications</b>	None. In child patterns.

## ListCommandNameChar

Capital Letters:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Small Letters:	a b c d e f g h i j k l m n o p q r s t u v w x y z
Digits:	1 2 3 4 5 6 7 8 9 0
Specials:	, . < ( *     & ! \$ * ) ; ' - / \ % _ > ? = " : # @ ~ \ ^ _ {

Figure 88. ListCommandNameChar

<b>Purpose</b>	<i>ListCommandChar</i> contains characters that are possible input values to a list's input fields.
<b>Partial Information Sequence</b>	<u>S</u> -Save <u>D</u> -Delete

<b>Representation</b>	Not Applicable.
<b>Modifications</b>	Additional characters.

## ListCommandTerminator

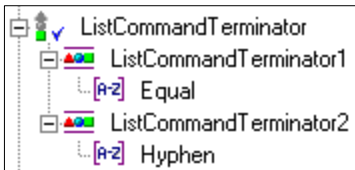


Figure 89. ListCommandTerminator

<b>Purpose</b>	The pattern <i>ListCommandTerminator</i> recognizes the characters that separate between the allowed values and the description of their meanings in the prompt for the allowed values of a list's input fields.
<b>Partial Information Sequence</b>	1-Update    2-Delete    3-View
<b>Representation</b>	None.
<b>Modifications</b>	New characters can be added as child patterns. <i>ListCommandTerminator</i> cannot have characters in common with <i>ListCommandChar</i> .

## ListHeadersHardDelimiter

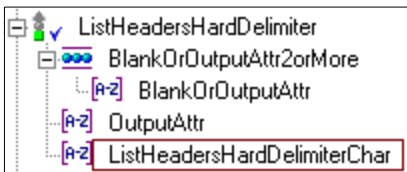


Figure 90. ListHeadersHardDelimiter

**Structure** The pattern *ListHeadersHardDelimiterChar* contains the characters:



**Purpose** The pattern *ListHeadersHardDelimiter* contains the character sequences that ACE **always** uses to divide the header row of a list into the headers of the individual columns. Character sequences that satisfy *ListHeadersHardDelimiter* are never available to be recognized as part of a column's header. The pattern will also typical "framing" characters from being recognized as a header sequence.

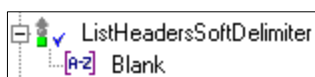
**Partial Information Sequence**

```
Time Stamp Size__Color +++
à0000000àà0000000à ò000000à
à0000000àà0000000à ò000000à
à0000000àà0000000à ò000000à
```

**Representation** None.

**Modifications** Not recommended.

## ListHeadersSoftDelimiter



**Figure 91. ListHeadersSoftDelimiter**

**Purpose** The pattern *ListHeadersSoftDelimiter* contains the character sequences that ACE optionally uses to divide the header row of a list into the headers of the individual columns. See *ListHeadersHardDelimiter*.

**Partial Information Sequence**

```
Time Stamp Size Color +++
à0000000àà0000000à ò000000à
à0000000àà0000000à ò000000à
à0000000àà0000000à ò000000à
```

**Representation** None.

**Modifications** Not recommended.

## ListLines

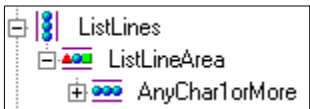


Figure 92. ListLines

<b>Purpose</b>	The pattern <i>ListLines</i> is a child pattern of various <i>List*</i> patterns. It is used by these <i>List*</i> patterns to determine the extent of a list on the host screen.
<b>Partial Information Sequence</b>	à00000à    à0000à à00000à    à0000à à00000à    à0000à
<b>Representation</b>	Forbidden.
<b>Modifications</b>	Forbidden.

## ListMoreIndicator

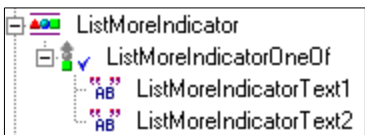
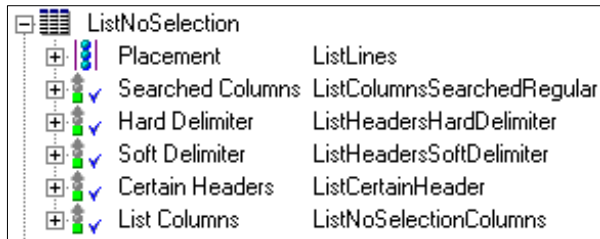


Figure 93. ListMoreIndicator

<b>Purpose</b>	The pattern <i>ListMoreIndicator</i> is used to recognize and ignore strings that indicate that a list contains too many rows to fit onto a single screen.
<b>Partial Information Sequence</b>	àmore...à

<b>Representation</b>	None. No representation should be necessary.
<b>Modifications</b>	Additional strings, for languages other than English.

## ListNoSelection



**Figure 94. ListNoSelection**

<b>Purpose</b>	<p>The pattern <i>ListNoSelection</i> presents a column of list input fields as a column of the GUI Table. The pattern does not attach GeneralUTMethod <i>DoubleClickTable</i> to the Table.</p> <p>The List Columns pattern <i>ListNoSelectionColumns</i> does not contain <i>ListLineCommand</i>. Thus, a column of list input fields is recognized by one of the other List Columns patterns, and the column of list input fields is represented by a Table column on the GUI.</p>
<b>Complete Information Sequence</b>	<p><u>àSexààNameààAgeà</u>  <u>àÀBàÀBBBBàÀBBBà</u>  <u>ÀBàÀBBBBàÀBBBà</u>  <u>ÀBàÀBBBBàÀBBBà</u></p>
<b>Representation</b>	Table.
<b>Modifications</b>	In child patterns.



# ListNotSubfile

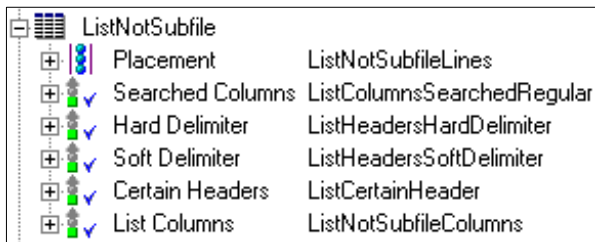


Figure 95. ListNotSubfile

## Purpose

The pattern *ListNotSubfile* recognizes characters arranged into rows and columns and delimited by attributes. The pattern is used to recognize characters that are functionally lists when there is no DDS information.

When you apply the *ListNotSubfile* **section** to the host screen, you must make sure that the section does not extend beyond the list.

## Complete Information Sequence

àNameààSizeà àNumà  
àÀBBBBàà0000à ò000à  
ÀBBBBàà0000à ò000à  
ÀBBBBàà0000à ò000à

## Representation

Table.

## Modifications

In child patterns.

# ListWithoutHeader



Figure 96. ListWithoutHeader

<b>Purpose</b>	The pattern <i>ListWithoutHeader</i> recognizes a list that does not have headers.
<b>Complete Information Sequence</b>	<pre> àÀBBBBàà0000à 0000à   ÀBBBBàà0000à 0000à   ÀBBBBàà0000à 0000à </pre>
<b>Representation</b>	Table, GeneralUTMethod.
<b>Modifications</b>	None In child patterns.

## Menu

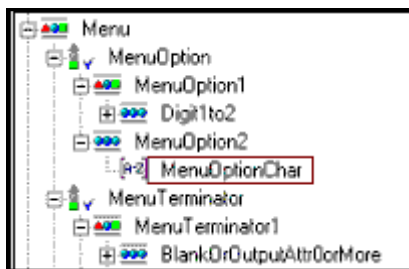


Figure 97. Menu

<b>Structure</b>	The pattern <i>MenuOptionChar</i> contains the characters:
------------------	--

<b>Capital Letters:</b>	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
<b>Small Letters:</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z

<b>Purpose</b>	The pattern <i>Menu</i> is used to recognize the complete prompt—input characters plus description—for one of the allowed inputs to a menu command field.
<b>Complete Information Sequence</b>	<pre> à 1. Manufacturing à 2. Finance à 3. Administration à 4. Distribution </pre>

<b>Representation</b>	PushButtons with the descriptive part of each prompt as text on the Button. GeneralUTMethod <i>SelectMenuOption</i> .
<b>Modifications</b>	None. See child patterns.

## MenuCommand

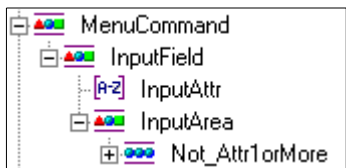
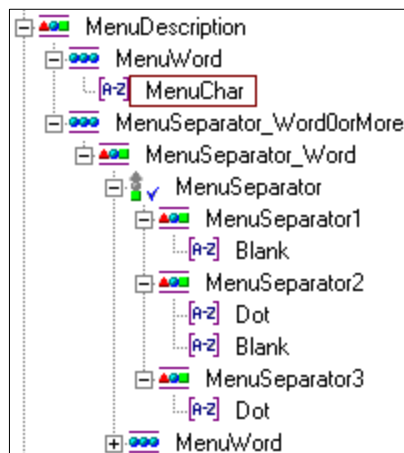


Figure 98. MenuCommand

<b>Purpose</b>	<p>The pattern <i>MenuCommand</i> is used to recognize an input field associated with a menu. Such an input field should have no direct representation on the GUI interface. The indirect representation on the interface is via the menu's PushButtons. Menu input fields are recognized by <i>MenuCommand</i> because <i>MenuCommand</i> is the only pattern containing <i>InputField</i> in the <i>Menu</i> Primary Section.</p> <p>If a second input field happens to lie within the area covered by the <i>Menu</i> Primary Section, apply an appropriate section by hand to the second input field.</p>
<b>Partial Information Sequence</b>	<pre> àEnter option numberà à==&gt;ÀBBà </pre>
<b>Representation</b>	A variable named "Command".
<b>Modifications</b>	None.

## MenuDescription



### Figure 99. MenuDescription

## Structure

The pattern *MenuChar* contains the characters:

Capital Letters:	A B C D E F G H I J K L M N O P Q R S T U V
Small Letters:	a b c d e f g h i j k l m n o p q r s t u v
Digits:	1 2 3 4 5 6 7 8 9 0
Specials:	, . < ( +   & ! \$ % * ) ; ' - / ` % _ ? = _ : # @ ~ \ ^ ' ,
Input Attributes:	■ ■ ■ ■ ■ ■ ■ ■ ' ■ ■ ■ ■ ■ ■ ■ A A A E E E İ İ Ø 0 0 0 Ø Ü Ü Þ
Non Input Attributes:	ç š ĩ ¨ ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ à á â ã ä å æ ç è é
Null:	■

## Purpose

The pattern *MenuDescription* is used to recognize the descriptive part of a prompt for the allowed inputs to a field.

## Partial Information Sequence

```
: à 1. Manufacturing
à 2. Finance à 3. Administration
à 4. Distribution
```

## Representation

None.

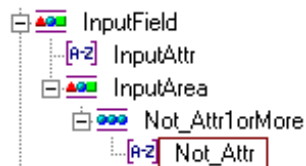
## Modifications

None. See child patterns. *MenuChar* and *MenuSeparator* cannot have characters in common.



**Purpose**

Character Set Pattern *Not\_Attr* is used to determine the extent of host screen input/output areas designated by an immediately preceding attribute character:



The field is recognized as extending until the next attribute character.

**Partial Information Sequence**

ÀBBBBÀ

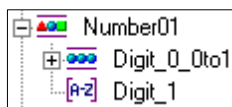
**Representation**

None.

**Modifications**

Forbidden.

## Number01



**Figure 102. Number01**

**Purpose**

The pattern *Number01* is the subpattern that limits *FKey01* to recognizing character sequences associated with the **first** command key.

**Partial Information Sequence**

F01=Previous Menu

**Representation**

None.

**Modifications**

None. Modifications can be made to *FKeyPrefix* or *FKeyTerminator*.

## OutputCheckBoxViaINIField

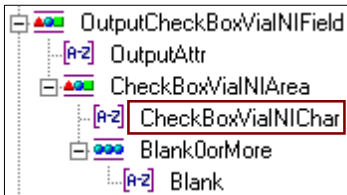
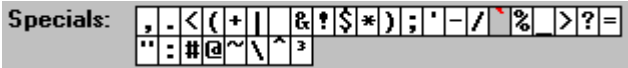


Figure 103. OutputCheckBoxViaINIField

**Structure** The Character Set *CheckBoxViaINIChar* contains a single character:



**Purpose** The Pattern *OutputCheckBoxViaINIField* is used in conjunction with customized screen maps and the CHECKBOX.INI file to identify a protected checkbox by means of a particular screen character. You use this mechanism when the host screen does not contain Yes/No or other characters that identify an output field as displaying only “yes or no.”

**Partial Information Sequence** àRead Only à`

**Representation** CheckBox

**Modifications** None. Change the identifier character in *CheckBoxViaINIChar*

## OutputComboBoxIdentifierFirstLine

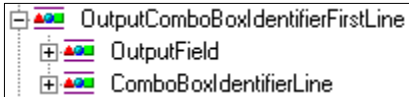


Figure 104. OutputComboBoxIdentifierFirstLine

<b>Purpose</b>	The pattern <i>OutputComboBoxIdentifierFirstLine</i> is used to allow for the possibility of more than one line of <i>EntryPairs</i> prompts on the host. The first line of prompts contains a protected field, while any following lines of prompts do not contain a field. Therefore, the <i>EntryPairs</i> in the following lines might not be aligned with the <i>EntryPairs</i> in the first line, and a vertical iteration of <b>all</b> the lines of <i>EntryPairs</i> is not possible.
<b>Partial Information Sequence</b>	<code>àOrder By:àBàà (1=Number, 2=Last _Name)à</code>
<b>Representation</b>	None. Representations are attached to the child patterns.
<b>Modifications</b>	None. In the child patterns.

## OutputComboBoxViaINIField

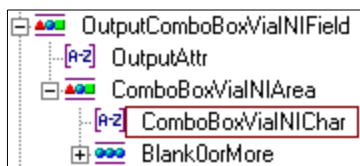


Figure 105. OutputComboBoxViaINIField

<b>Structure</b>	The Character Set <i>ComboBoxViaINIChar</i> contains a single character																																										
	<div><div>Specials:</div><table><tr><td>,</td><td>.</td><td>&lt;</td><td>(</td><td>+</td><td> </td><td>&amp;</td><td>!</td><td>\$</td><td>*</td><td>)</td><td>;</td><td>'</td><td>-</td><td>/</td><td>`</td><td>%</td><td>_</td><td>&gt;</td><td>?</td><td>=</td></tr><tr><td>"</td><td>:</td><td>#</td><td>@</td><td>~</td><td>\</td><td>^</td><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></div>	,	.	<	(	+		&	!	\$	*	)	;	'	-	/	`	%	_	>	?	=	"	:	#	@	~	\	^	3													
,	.	<	(	+		&	!	\$	*	)	;	'	-	/	`	%	_	>	?	=																							
"	:	#	@	~	\	^	3																																				
<b>Purpose</b>	The Pattern <i>OutputComboBoxViaINIField</i> is used in conjunction with customized screen maps and the COMBOBOX.INI file to identify a protected combo by means of a particular screen character. You use this mechanism when an output field has a limited number of allowed values, and the host screen does not contain a list of these allowed values.																																										



<b>Partial Information Sequence</b>	<code>àStatus:à    <u>à\</u></code>
<b>Representation</b>	ComboBox
<b>Modifications</b>	None. Change the identifier character in <i>ComboBoxViaINIChar</i>

## SystemIDOutputField

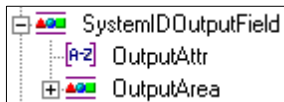


Figure 106. SystemIDOutputField

<b>Purpose</b>	The pattern <i>SystemIDOutputField</i> is used to recognize non DDS SystemIDs by <b>assuming</b> that SystemIDs appear on the top right hand corner of host screens.
<b>Complete Information Sequence</b>	<code><u>àSystemID...à</u></code>
<b>Representation</b>	Variable.
<b>Modifications</b>	Change location. Add a GUI representation such as a Static control or a Variable.

## SystemIDViaSDFArea

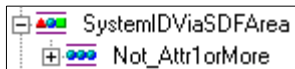
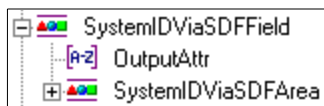


Figure 107. SystemIDViaSDFArea

<b>Purpose</b>	The pattern <i>SystemIDViaSDFArea</i> is the sole member of DDS filter section <i>SysName</i> . This ensures that <i>SystemIDViaSDFArea</i> is the only pattern that recognizes DDS System ID's.
<b>Partial Information Sequence</b>	<u>àSystemID...</u> à
<b>Representation</b>	None.
<b>Modifications</b>	None.

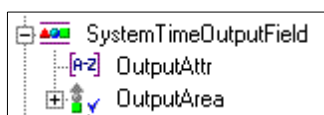
## SystemIDViaSDFField



**Figure 108. SystemIDViaSDFField**

<b>Purpose</b>	The pattern <i>SystemIDViaSDFField</i> is the only Primary Pattern that recognizes DDS System ID's.
<b>Complete Information Sequence</b>	<u>àSystemID...</u> à
<b>Representation</b>	Variable.
<b>Modifications</b>	Add a GUI representation such as a Static control or a Variable.

## SystemTimeOutputField



**Figure 109. SystemTimeOutputField**

<b>Purpose</b>	The pattern <i>SystemTimeOutputField</i> is used to recognize non DDS Time's by <b>assuming</b> that the Time output appears on the top right hand corner of host screens.
<b>Complete Information Sequence</b>	<u>àTime...à</u>
<b>Representation</b>	None.
<b>Modifications</b>	Change location. Add a GUI representation such as a Static control or a Variable.

## SystemTimeViaSDFArea

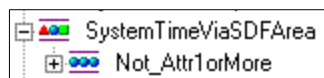


Figure 110. SystemTimeViaSDFArea

<b>Purpose</b>	The pattern <i>SystemTimeViaSDFArea</i> is <i>designed</i> to be the sole member of DDS filter section <i>Time</i> . This can ensure that <i>SystemTimeViaSDFArea</i> is the only pattern that recognizes DDS Time. At present, the pattern <i>IgnoreAny</i> precedes <i>SystemTimeViaSSDFArea</i> in filter section <i>Time</i> . This means that by default a DDS Time is ignored by ACE.
<b>Partial Information Sequence</b>	<u>àTime...à</u>
<b>Representation</b>	None.
<b>Modifications</b>	Remove <i>IgnoreAny</i> from DDS filter Section <i>Time</i> in order to activate <i>SystemTimeViaSDFArea</i> .

## SystemTimeViaSDFField

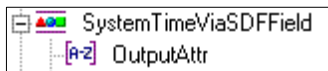


Figure 111. `SystemTimeViaSDFField`

<b>Purpose</b>	The pattern <i>SystemTimeViaSDFField</i> is <i>designed</i> to be the only Primary Pattern that recognizes DDS Time's.
<b>Complete Information Sequence</b>	<u>àTime...à</u>
<b>Representation</b>	None.
<b>Modifications</b>	Add a GUI representation such as a Static control or a Variable.

## UserIDOutputField

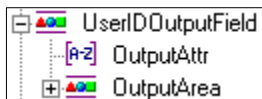


Figure 112. `UserIDOutputField`

<b>Purpose</b>	The pattern <i>UserIDOutputField</i> is used to recognize non DDS UserIDs by <b>assuming</b> that UserIDs appear on the top left hand corner of host screens.
<b>Complete Information Sequence</b>	<u>àUserID...à</u>
<b>Representation</b>	Variable.
<b>Modifications</b>	Change location. Add a GUI representation such as a Static control or a MenuItem/MsgBox combination.

# UserIDViaSDFArea

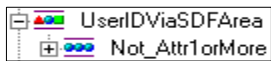


Figure 113. UserIDViaSDFArea

<b>Purpose</b>	The pattern <i>UserIDViaSDFArea</i> is the sole member of DDS filter section <i>User</i> . This ensures that <i>UserIDViaSDFArea</i> is the only pattern that recognizes DDS User ID's.
<b>Partial Information Sequence</b>	<u>àUserID...</u> à
<b>Representation</b>	None.
<b>Modifications</b>	None.

# UserIDViaSDFField

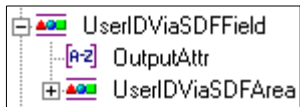


Figure 114. UserIDViaSDFField

<b>Purpose</b>	The pattern <i>UserIDViaSDFField</i> is the only Primary Pattern that recognizes DDS User ID's.
<b>Complete Information Sequence</b>	<u>àUserID...</u> à
<b>Representation</b>	Variable.
<b>Modifications</b>	Add a GUI representation such as a Static control or a MenuItem/MsgBox combination.



---

# Index

## Symbols

.INI  
files  
and controls 111

## C

Character set 52  
Check box  
pattern definitions 73  
Combo box  
pattern definitions 79  
Controls  
Via INI files 111

## D

Dynamic  
pattern definitions 86

## F

Filters and pattern definitions 114

## H

Host  
HostBasedFormats 88, 95, 96

## K

KnowledgeBase  
main principle of 41  
modifications  
best method of 51  
consistency of 65  
new pattern definitions 59  
using case 56

## L

Lists 104  
List delimiters 101

Local modifications 26  
Location  
location and pattern definitions 114

## O

OneOf 51, 62

## P

Pattern definitions  
and character sequences 44  
and filters 114  
and location 114  
character set 52  
dynamic 86  
lists 104  
OneOf 51, 62  
search order 61  
strings 103  
Priority  
of pattern definitions 61

## R

Representation component  
check box 73  
combo box 79  
HostBasedFormats 88, 95, 96  
table 104

## S

Sections  
ApplyToAll 84  
search order of pattern definitions 61  
Strings 103

## T

Tables 104

