

# InData

**Version 2.0**



# InData™ 2.0 User's Guide

Em Software, Inc.  
December, 2002

## Keeping in Touch with Em via the Web

Em maintains a World-Wide Web (Internet) site at [www.emsoftware.com](http://www.emsoftware.com), with up-to-date information about InData and its other products. We will provide news of major updates, minor version software updaters, free software, on-line tips and techniques to augment this manual, and the like, at this site, so we encourage you to check it every so often.

## Technical Support

If you have problems using InData, please first consult the “Troubleshooting and Error Messages” chapter in this manual to find many common problems and their solutions—your answer is very likely to be found there, and you can save yourself a lot of trouble with just a little reading. For additional information, consult the frequently-asked questions and other usage hints on the support pages of Em’s web site at <http://www.emsoftware.com/support>.

## Contact Information

Em Software, Inc.  
503 Belleview Blvd.  
Steubenville, Ohio 43952 USA  
**vox** 740 284 1010 (main), 877 984 1010 (sales)  
**fax** 740 284 1210 (main, sales)  
**web** [www.emsoftware.com](http://www.emsoftware.com)  
**email** [support@emsoftware.com](mailto:support@emsoftware.com) or [info@emsoftware.com](mailto:info@emsoftware.com)

## **Copyright and Trademark Information**

This manual and software are Copyright © 2000-2003, Em Software, Inc. All rights reserved.

InData and InFlow are trademarks of Em Software. All other trademarks and registered trademarks are property of their respective holders.

## **Credits**

This manual was written by Æleen Frisch, Exponential Consulting, North Haven, Connecticut. Production was done by Æleen Frisch and Cat Dubail. The original (Xdata) software was conceived and implemented by Chris Ryland, Em Software (AMDG). The InDesign rewrite was performed by Chris Roueche. The Em logo (on the manual cover) was created by A&M Design, Madison, Connecticut.

## **Version Information**

Manual version 2.0, corresponds to InData version 2.0, December, 2002.

# Table of Contents

List of Tips and Techniques .....	xi
<b>1 Introduction .....</b>	<b>1</b>
System Requirements .....	1
What You Need to Know ... ..	2
<i>About Your Computer</i> .....	2
<i>About InDesign</i> .....	2
<i>About Your Database or Spreadsheet Program</i> .....	2
About This Manual .....	2
<i>InData and InDesign Versions</i> .....	3
<i>Platform-Specific Issues</i> .....	4
<i>Typographical Conventions</i> .....	4
<b>2 Installing InData .....</b>	<b>7</b>
InData Distribution Contents .....	7
<i>About InFlow</i> .....	8
Installing the InData Plug-ins .....	9
<i>Installing InData's Auxiliary Files</i> .....	9
Personalizing Your Copy of InData .....	9
<b>3 InData Tutorials .....</b>	<b>11</b>
Preliminary Information .....	11
Tutorial 1: Preparing an Address List .....	12
<i>Make Sure InData is Installed Properly</i> .....	12
<i>Open the Tutorial File</i> .....	12
<i>Examine the Document Setup</i> .....	12
<i>Examine the InData Prototype</i> .....	14
<i>Look at the Prototype's Paragraph Attributes</i> .....	16
<i>Import Data into the Document</i> .....	17
<i>Undoing a Data Import</i> .....	20
<i>View During Import Options</i> .....	20
Tutorial 2: Creating a Company Phone List .....	21
<i>Setting up the Document</i> .....	21
<i>Placing Fixed Text and Graphics on the Master Page</i> .....	21
<i>Creating the InData Prototype</i> .....	22
<i>Setting InData Preferences</i> .....	25
<i>Importing Data into the Document</i> .....	26
<i>Making a Deliberate Mistake</i> .....	26
<i>On Your Own</i> .....	27
Tutorial 3: Advanced Data Importing .....	27
<i>Adding a Title on the First Page</i> .....	28
<i>Avoiding Unwanted Blank Lines From Missing Fields</i> .....	29
<i>Adding Room Numbers</i> .....	30

<i>Import the Data</i> .....	31
Tutorial 4: Adding Department Headlines .....	32
<i>Comparing the Current and Previous Records</i> .....	32
<i>Creating Reversed Type with Paragraph Rules</i> .....	34
<i>Forcing a Paragraph to the Top of a Column</i> .....	35
Tutorial 5: Importing Pictures .....	36
<i>Adding a Picture Frame to the Prototype</i> .....	36
<i>Specifying the Picture Filename in the Prototype</i> .....	37
<i>Making the Picture Import Conditional</i> .....	38
<i>Specifying Global Picture Attributes</i> .....	38
<i>Import the Data</i> .....	39
<i>When InData Can't Find a Picture File</i> .....	40
Tutorial 6: Adding Document Headers and Footers .....	40
<i>Creating a Mark in the Prototype</i> .....	41
<i>Adding a Mark Reference to the Master Pages</i> .....	42
<i>Informing InData About the Mark Reference</i> .....	42
<i>Save the Document as a Template</i> .....	44
<i>Import the Data</i> .....	44
Tutorial 7: Automated Document Creation .....	45
<i>Macintosh Scripting: AppleScript</i> .....	45
<i>Windows Scripting: VBScript</i> .....	46
<i>Preparing a Document for Automated Data Importing</i> .....	47
<i>Naming Stories</i> .....	48
<i>Macintosh: Prepare the AppleScript</i> .....	49
<i>Windows: Prepare the VBScript</i> .....	50
<i>Perform the Automated Import Operations</i> .....	51
<b>4 Basic InData Operations</b> .....	<b>53</b>
A Quick Overview of InData .....	53
<i>Prepare the InDesign Document</i> .....	53
<i>Create an InData Prototype</i> .....	55
<i>Specifying the Formatting of Incoming Data</i> .....	56
<i>Designating the Prototype</i> .....	57
<i>Importing Data</i> .....	57
InData Prototype Fundamentals .....	58
<i>Where to Put the Prototype</i> .....	58
<i>Selecting Field Names</i> .....	61
<i>Skipping One or More Fields in the Data File</i> .....	62
<i>Inserting Fields More Than Once</i> .....	62
<i>Field Placement Flexibility</i> .....	63
<i>Missing and Empty Fields</i> .....	63
<i>When to Omit the Right Chevron Mark</i> .....	63
More Examples of Formatting Prototypes .....	64
<i>Forcing Text to the Next Column, Frame, Page and Even/Odd Page</i> ..	66
The InData Menu .....	67
Data Import Options .....	68
<i>Importing a Range of Records</i> .....	71
<i>Document View Options</i> .....	71

<i>Controlling Data Importing</i> .....	72
<i>Prototype Errors</i> .....	73
Setting InData Preferences .....	74
<b>5 Preparing Data for Importing</b> .....	<b>77</b>
Data File Formats .....	78
<i>Handling Carriage Returns within Fields</i> .....	78
General Exporting Procedures .....	78
Exporting Data from Database Applications:	
An Example from FileMaker Pro .....	79
<i>Handling FileMaker Pro Databases with Repeating Fields</i> .....	81
Exporting Data from Database Applications:	
An Example from Visual FoxPro .....	81
Exporting Data from Spreadsheet Applications:	
An Example from Excel .....	83
Creating Data Files Manually .....	84
<b>6 Conditional Data Importing</b> .....	<b>85</b>
if Statements .....	86
Constructing Conditions .....	87
<i>Some Example Conditional Prototype Statements</i> .....	89
<i>if...else Chains</i> .....	90
<i>Alternate Forms of the if Statement</i> .....	92
<i>Forming Compound Conditions</i> .....	92
<i>Arithmetic Operators</i> .....	93
<i>Nested if Statements</i> .....	93
Comparing with the Previous or Next Record .....	94
<i>Avoiding Unwanted Blank Lines and Empty Text Boxes</i> .....	94
Doing Something Every nth Record .....	96
Constructing and Using Loops within Prototypes .....	99
<i>Other Forms of the repeat Statement</i> .....	99
<i>Leaving a Loop Early</i> .....	101
<i>Ending a Loop Iteration Early</i> .....	101
<i>Using Picture Frames in a Loop</i> .....	102
<b>7 Manipulating Incoming Data</b> .....	<b>103</b>
Extracting Parts of Fields and Expressions .....	103
<i>Finding the Length of a Character String</i> .....	106
<i>Extracting Substrings</i> .....	107
<i>String Concatenation</i> .....	107
<i>Including Literal Chevron Marks in a Prototype</i> .....	108
Extracting Words and Lines from Expressions .....	108
<i>Extracting Lines</i> .....	109
<i>Extracting Arbitrary Items</i> .....	110
<i>Determining the Number of Chunks</i> .....	110
Handling Repeating Fields .....	111
String Conversion Functions .....	112
Determining the Current Record or Page Number .....	114

Accessing Arbitrary Fields within Records .....	115
<b>8 Importing and Formatting Pictures .....</b>	<b>117</b>
Importing Pictures .....	117
<i>Importing Pictures Conditionally</i> .....	119
Setting Picture Frame Attributes .....	121
<i>Prototype Statements for Setting Picture Frame Attributes</i> .....	122
Setting Default Directory Locations for Picture Files .....	124
Handling Missing Picture Files .....	125
Specifying Precise Picture Locations on the Page .....	125
<b>9 Controlling Document Layout .....</b>	<b>127</b>
Creating a Mark in the Prototype .....	127
Adding a Mark Reference to the Master Pages .....	128
Updating Existing Headers and Footers .....	130
<i>Changing the Headers and Footers Themselves</i> .....	131
Applying Master Pages within a Prototype .....	132
<b>10 Advanced Prototypes .....</b>	<b>135</b>
Record Input Control Statements .....	135
Setting Variables .....	137
<i>Manually Wrapping Text Columns by Words</i> .....	138
<i>Implementing Fixed-Width Fields</i> .....	140
Testing the Data Type of Expressions .....	140
Soliciting Input at Import Time .....	141
<i>Asking More than One Question</i> .....	141
<i>Asking Only Once</i> .....	141
Using Multiple fields Statements .....	142
InDesign Tags Support .....	143
<i>More Complex put styled Statements</i> .....	144
<i>Inserting an Entire File of Text</i> .....	145
<b>11 Hints for Debugging Prototypes .....</b>	<b>147</b>
Test with a Few Sample Records First .....	147
Build up the Prototype Gradually .....	147
Use Multiple Text Frames for the Prototype .....	149
Make Sure the Data is OK .....	150
Downplaying Prototype Statements .....	150
<b>12 Automating Document Building .....</b>	<b>153</b>
Conceptual Overview .....	153
Naming Stories .....	154
<i>Naming Substories</i> .....	155
<i>Finding Stories and Substories by Name</i> .....	155
<i>Building Complex Multi-Part Documents</i> .....	156
Using InData with AppleScript (Macintosh) .....	156
<i>An Example Script</i> .....	157
<i>Importing Large Data Files in Batches</i> .....	157



---

Using InData with Visual Basic (Windows)	158
<i>An Example Script</i>	159
<b>13 InData Reference</b>	<b>161</b>
Entering Special Characters	161
The InData Menu	162
<i>InData Preferences Submenu</i>	163
InData Control Panel Buttons	163
Data Preferences Dialog	164
View Preferences Dialog	165
Find Story/Substory Dialog	165
General Preferences Dialog	166
Make Header/Footer Dialog	167
Name Story and Name Substory Dialogs	167
Range Preferences Dialog	168
InData Prototype Elements	168
<i>Prototype Statements</i>	169
<i>Expression Operators</i>	174
<i>Built-In Constants and Variables</i>	179
<i>Integer Operations</i>	180
<i>Comparison Operations</i>	180
<i>Logical Operations</i>	181
<i>Grouping Operations</i>	181
InData Technical Information	181
<i>InData Limitations</i>	181
<i>InData Memory Usage (Macintosh)</i>	181
<i>InData Reserved Words</i>	182
<i>Additional Prototype Restrictions</i>	182
<i>Processing Very Large Data Files</i>	183
<i>Page Complexity</i>	183
<b>14 Troubleshooting and Error Messages</b>	<b>185</b>
Common Problems and Their Causes	185
Status Messages	187
Error Messages	189
<i>User Interface Errors</i>	190
<i>Prototype Structure Errors</i>	192
<i>Prototype Execution Errors</i>	195
<i>AppleEvent Scripting Errors</i>	200
<b>Index</b>	<b>201</b>

---

---

## List of Tips and Techniques

Adding page numbers to a document . . . . .	22
Applying master pages automatically . . . . .	132
Avoiding empty text boxes . . . . .	94
Avoiding unwanted blank lines . . . . .	29-30
Capitalizing/uncapitalizing character strings . . . . .	112
Converting quotes and double dashed automatically . . . . .	144
Creating multiple output records for each incoming record . . . . .	99
Creating reversed type . . . . .	34
Creating running headers and footers . . . . .	127
Creating shaded and ruled tables . . . . .	97-98
Determining whether a file exists . . . . .	145
Doing Something Every N Records . . . . .	96, 135
Forcing a paragraph to the top of a column/page . . . . .	35
Formatting fractions . . . . .	113
Formatting phone numbers . . . . .	104
Formatting prices . . . . .	104-105
Handling fixed-width fields . . . . .	140
Including pictures within text . . . . .	37
Inserting a text file within the imported data . . . . .	145
Linking text boxes into the automatic text chain . . . . .	54
Performing different actions based on a field value . . . . .	90
Positioning picture boxes precisely . . . . .	125
Presetting InData dialog settings . . . . .	174
Prompting for input during imports . . . . .	141
Saving documents as templates . . . . .	26
Translating numerically coded fields to descriptive strings . . . . .	108-109
Updating Imported Data . . . . .	57
Wrapping text by words . . . . .	138



# 1

## Introduction

InData™ is a sophisticated data publishing facility for use with Adobe InDesign.® It automates the formatting of documents containing repeating data units such as catalogs, directories, mailing labels, price lists, and schedules, supporting both text and graphics. Formerly, each separate element in such documents had to be formatted by hand. InData automatically applies the format you specify to each element of information as it is imported from the original data source. InData accepts data in the major Macintosh and Windows data exchange file formats, enabling you to use it with textual data created in all major database and spreadsheet packages, or even with data created by hand in a word processor.

InData operates as an plug-in to InDesign, automatically becoming part of the InDesign publishing system once it is installed. InData runs from inside of InDesign, under its own menu, and imports and formats text into normal InDesign publications. It is not a stand-alone program which must be run separately from InDesign, and no special additional files are required. (See the InDesign reference manual for general information about plug-ins.)

InData is available in both a MacOS (PowerPC) version and in a Windows (Intel) version. The Macintosh version is fully carbonized and thus runs natively under both Mac OS 9 and Mac OS X.

Since InDesign doesn't have the notion of auto-flow (automatic page creation) except when placing text, we provide the InFlow plug-in as a companion to InData to provide this functionality.

## System Requirements

In order to run InData, you will need a Macintosh or Windows-based computer running InDesign. The computer will need sufficient memory and hard disk space to support InDesign (consult the InDesign documentation for details). MacOS-based computers should run Mac OS 9 or Mac OS X. Windows 98, NT, 2000, ME and XP are supported on Intel systems (to the extent that InDesign itself supports each operating system).

## What You Need to Know ...

### About Your Computer

You should be familiar with basic Macintosh or Windows concepts and procedures, such as using the mouse, selecting items from menus, entering information in dialogs, navigating among folders, and manipulating files (e.g. copying, renaming, and deleting).

### About InDesign

You should be comfortable with basic InDesign tasks. You should know how to:

- ◆ Specify text formats: font, size, style, and so on.
- ◆ Set paragraph formats, including indents, before and after spacing, tabs and rules.
- ◆ Work with text and picture frames.
- ◆ Use the rulers and guides.
- ◆ Edit text inside InDesign.
- ◆ Set up and assign master pages.
- ◆ Save and print publications.

It is very helpful, but not essential, to understand InDesign templates and character and paragraph styles.

### About Your Database or Spreadsheet Program

This manual generally assumes that you are familiar with designing and manipulating databases or spreadsheets with your database and spreadsheet applications and that you know the meanings of terms like *record* and *field*. You will also need to know how to export data from your application. Chapter 5, “Preparing Data for Importing,” reviews data exporting procedures for several popular applications, and discusses general considerations applicable to all such packages. However, be sure to consult the documentation for your own application for the details of its export procedure.

## About This Manual

Chapter 2, “Installing InData,” describes the steps necessary to install InData.

Chapter 3 contains the InData Tutorials. These seven tutorials provide a hands-on, step-by-step introduction to InData’s essential features, suitable for new users of InData. Experienced InDesign users may want to skip the early tutorials, or even the whole section, and begin with the “Quick Overview,” which makes up the first section of Chapter 4.

Chapter 4, “Basic InData Operations,” contains a quick overview of InData as its first section. Later sections discuss basic concepts and simple prototypes—the

mechanism for telling InData how to place and format imported data records—in more detail. This chapter also discusses InData menus and dialogs.

Chapter 5, “Preparing Data for Importing,” provides a general discussion of data export techniques from database and spreadsheet programs, using Excel, AppleWorks and FoxPro as examples.

Chapter 6, “Conditional Data Importing,” considers more advanced prototypes in which data is imported only when specified conditions are present. It also discusses creating loops within InData prototypes.

Chapter 7, “Manipulating Incoming Data,” discussing various techniques for extracting parts of and transformations of the raw data as it is imported.

Chapter 8, “Importing and Formatting Pictures,” covers all aspects of importing pictures with InData.

Chapter 9, “Controlling Document Layout,” discusses creating running headers and footers and assigning master pages within InData prototypes.

Chapter 10, “Advanced Prototypes,” discusses the more complex features of InData’s prototype language, including soliciting user input, controlling record processing, using variables in prototypes, and building formatted data using InDesign Tags.

Chapter 11, “Hints for Debugging Prototypes,” discusses techniques for building complex prototypes and sorting out the bugs in any prototype.

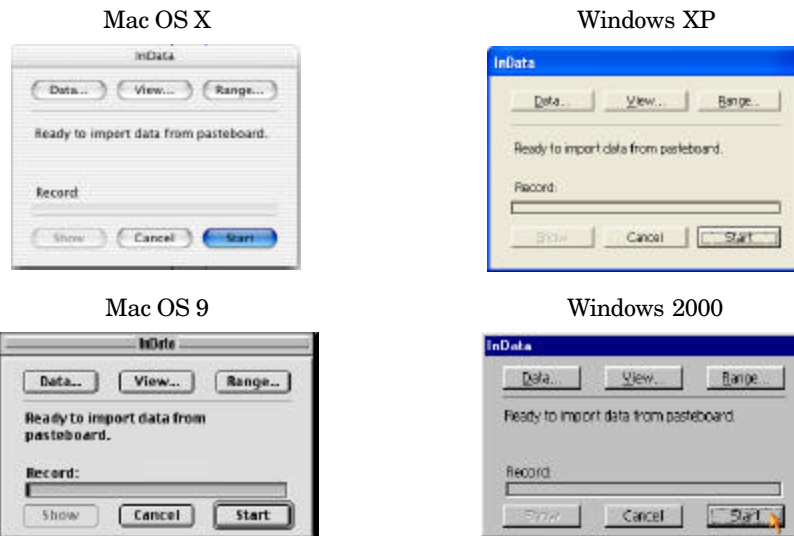
Chapter 12, “Automating Document Building,” discusses methods for automating InData operations using the AppleEvent scripting facility available under MacOS and Windows Automation using Visual Basic.

Chapter 13, “InData Reference,” contains descriptions of all InData menu items and dialog fields as well as a complete description of the InData prototype language. It also discusses various InData fine points, including its incoming data conversion and string comparison specifications.

Chapter 14, “Troubleshooting and Error Messages,” describes solutions to common InData importing problems and explains the InData error messages.

## Platform-Specific Issues

InData has been designed to work as similarly as possible on all of its supported computer and operating systems. Differences between versions will be noted as appropriate. Note, however, that the dialogs appear very similarly under the various operating systems. For example, here is the same sample dialog frame as it appears under Mac OS X (upper left), Mac OS 9 (lower left), Windows XP (upper right) and Windows 2000 (lower right):



Note that the majority of the screen shots in this manual are from Mac OS X or Windows XP systems, and we tend to alternate between them by page or section. However, a few shots from the older operating systems also still linger in this manual. As you can see from the preceding examples, however, the differences between the various interfaces are purely cosmetic.

Finally, in this manual, we will consistently refer to directories and subdirectories as “folders,” a term which is used in both Macintosh and Windows parlance.

## Typographical Conventions

We use alternate typefaces in this manual to distinguish certain types of items within the text. Generally, InData prototypes will be set off in separate paragraphs set in boldface Helvetica type, like this:

«fields last, first, middle, address, city, state, zip»

When we need to talk about prototype elements within a paragraph, we'll again set them in boldface Helvetica type. For example, the previous prototype illustrates an example of the **fields** statement.



When we are discussing prototype syntax, we will use italic Helvetica type to indicate general prototype elements (i.e., parameters within the prototype) which will need to be replaced by specific text when used in a InDesign document, as in this example:

«**fields** *new\_field*, **last**, **first**, **middle**, **address**, **city**, **state**, **zip**»

In this example, *new\_field* is a placeholder for a field name that you would provide when you actually used such a prototype. Notice that we set such parameters in italics when we use them within regular text as well.

When we add comments to prototypes for exposition only, we will set them in regular italic type to distinguish them from the prototype itself, as in this example:

«**if prev name <> name**» «**name**» *Insert only new names.*  
«**endif**»

The names of InDesign and InData menu items and dialog fields are set in bold-face type, as in “select **Open...** from the **File** menu” and “click the **Cancel** button to change your mind.” We use the following syntax to represent paths through a series of nested menus: the form **InData=>Preferences=>General...** means to select the **Preferences** item from the **InData** menu in InDesign and then to select the **General...** item from the **Preferences** “slideoff” submenu.

Finally, file and folder names within regular text are set in plain Helvetica type, as in “this file should be placed in the Plug-ins folder.”

---

# 2

## Installing InData

This chapter describes the steps needed to install InData.

### InData Distribution Contents

InData is distributed in electronic form (StuffIt archive under MacOS, WinZip file under Windows), usually from Em Software's web site. If you've purchased the product, you also should have received a serial number that will fully unlock it. These are the files you will find in the InData folder.

InData 2.0.pln

The InData plug-in file.

InFlow 2.0.pln

An auxiliary Em Software plug-in that enhances InDesign's default text frame threading capabilities (described in more detail in the next section).

Read Me

A text file containing last-minute notes, known problems with InData and InDesign, any system or plug-in conflicts, etc. We *highly recommend* you read this to avoid any surprises.



Samples

A folder containing demonstration files, including a Read Me text file explaining what each sample does and how to use it (so that the demonstration and release distributions can be identical).

Scripting Examples

A folder containing files illustrating ways of automating InData operations, including an explanatory Read Me text file. This folder contains a demo document, Demo Doc, and other text and picture files that it uses. The sample scripts themselves are stored in a subfolder called Visual Basic on Windows systems and called AppleScript on Macintosh systems. There may also be additional demonstration files present in this folder.

Tutorial

A folder containing the tutorial files for this InData manual.

There may be additional files in your InData folder. In this case, the Read Me file will describe their contents.

## About InFlow

The InFlow PlugIn adds the ability to mark arbitrary master page text thread frames as “autoflow.” When text in an autoflow thread becomes overset, a new page is added and the overset text is automatically flowed into the new page’s autoflow thread.

Note that InFlow is logically independent of InData and adds a valuable feature to InDesign. We went to some lengths to develop it, and are including it with InData, because the latter generally requires this feature in normal use. InFlow has a 10 page limit unless a registered copy of another Em Software product is present (e.g., InData or InCatalog).

QuarkXPress users should find InFlow very familiar, from the ability to create new documents with full-page autoflow text frames, to the ability to mark arbitrary master page text threads as autoflow.

With InFlow installed, marking the **Master Text Frame** checkbox in the **New Document** dialog creates a single full-page autoflow text frame on each of the initial master spread’s pages. This allows for easy creation of autoflow documents.

InFlow also adds a **Set Autoflow Thread** item to the **Object** menu. It allows the user to mark any master page’s text thread as an autoflow thread. If the selected master page frame is already the default autoflow thread on that page, then the menu item will change to **Clear Autoflow Thread**; this item can then be used to remove the autoflow property. Only empty master page text frames may be marked as autoflow, and all frames in that thread must be on a single page. Only one default autoflow thread per master page is allowed.

## Copy Protection

InData is serialized but, like InDesign, not network-copy-protected, so it will never complain about too many copies running at once. However, if you fail to purchase enough copies to cover the maximum simultaneous usage of InData at your site, you are legally in violation of your license agreement, and more importantly, ethically guilty of stealing from Em Software.

We have some of the most aggressive multi-pack discounts in the industry to make multiple copies affordable, to encourage you to stay legal as well as ethical.

## Installing the InData Plug-ins

Installing InData is very simple. Drag-copy the two plug-ins (InData 2.0.pln and InFlow 2.0.pln) from the distribution folder to your InDesign folder's Plug-ins subfolder, or, more appropriately, to one of the sub-sub-folders such as Filters. We recommend that you create a new subfolder named Em Software and that you place all relevant plug-in files there. Once the plug-in files are in place, InData is ready to use and will be available the next time InDesign is started. If InDesign is currently running, end the current session and restart it before trying to use InData.

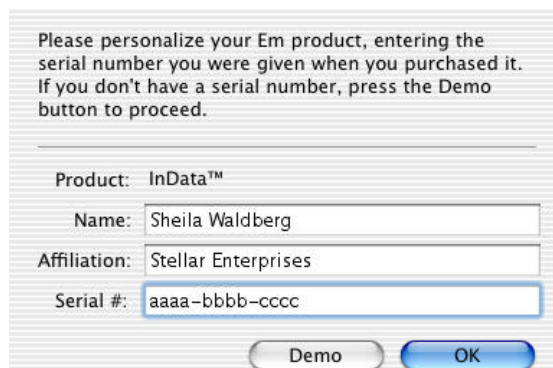
If you haven't already done so, fill out and email your registration data now (a form is provided in the top-level Read Me file). Registered InData users receive notices of upgrades, occasional special offers, etc.

## Installing InData's Auxiliary Files

You may also want to install the various InData auxiliary folders (listed in the "distribution contents" section above) somewhere. They may be copied to any folder you choose, or you can leave them in the distribution folder.

## Personalizing Your Copy of InData

The first time that you start up InDesign after installing InData, you will be prompted to fill in your name, organizational affiliation and InData serial number:



Please personalize your Em product, entering the serial number you were given when you purchased it. If you don't have a serial number, press the Demo button to proceed.

Product: InData™

Name: Sheila Waldberg

Affiliation: Stellar Enterprises

Serial #: aaaa-bbbb-cccc

Demo OK

Your InData serial number should have been provided over the phone, by email or via fax when you purchased InData. Be sure to record the serial number somewhere safely and permanently—if you lose it, we won't necessarily be able to find it for you.

Note that the serial number is stored in a file called InData.Reg in InData's containing folder. If you move InData to another machine, you should also move the InData.Reg file along with InData.pln unless you don't mind re-entering your seri-

al number. And, if you install later free upgrades for InData, you won't have to re-enter the serial number after installing InData, as it'll be remembered in this file.

If you're using InData in demonstration mode, you'll also get an InFlow serialization dialog, in which you should press the **Demo** button to proceed. (If you're using a serialized copy of InData, InFlow will automatically work without serialization, since it's bundled.)

# 3

## InData Tutorials

This section contains seven tutorials introducing InData. Each one should take you from fifteen to forty-five minutes. After completing these tutorials, you will be ready to begin importing and formatting data with InData in your own InDesign publications and to go on to the other chapters in this manual which present InData's advanced features.

### Preliminary Information

Using InData involves three elements:

- ◆ A ***raw data file***, containing the records to be imported and formatted using InData and InDesign, exported from a database or spreadsheet application, or prepared by hand using a word processing program. Generally, these records are stored in an external file, but they may also be imported from the clipboard or from a text story on the InDesign pasteboard.
- ◆ A ***InDesign document***, either designed from scratch or created by opening a template.
- ◆ An ***InData prototype***, which tells InData how the imported records are to be placed and formatted. As we'll see, the prototype is not a separate file, but rather exists as text within the InDesign publication.

These components will be explored in detail in the tutorials that follow.

All of the tutorials will assume that you have already started the InDesign program and that the tutorial files have been installed onto your system somewhere (see Chapter 2).

## Tutorial 1: Preparing an Address List

In this tutorial, you will examine an InData prototype and use it to import and format a company address and phone list.

### Make Sure InData is Installed Properly

InDesign should have a new menu named **InData** to the right of the **Object** menu.



This menu is used to perform the various InData operations, and you will use it throughout these tutorials.

If no **InData** menu is present, then InData is not installed properly. Check to make sure that the InData.pln file is stored in the InDesign Plug-ins folder or one of its sub-folders, and that you have restarted InDesign since placing it there.

### Open the Tutorial File

- 1 Choose **Open** from the **File** menu and then select the folder containing the InData tutorials files.
- 2 Once the tutorials folder is open, select the file Tutor\_1, make sure that **Open As** is set to **Normal**, and then click **OK** to open it. The file Tutor\_1 is a InDesign template; opening it in this way creates a new file with its same format while leaving the original unchanged. The new file will eventually become the Odin Mining Company's employee address and phone directory.

### Examine the Document Setup

The Tutor\_1 document is all ready to import data with InData. We will use it to illustrate the use of InData prototypes within InDesign documents and to demonstrate the data importing process.



When you open the document, its first (and only) page will be positioned in the InDesign document window:



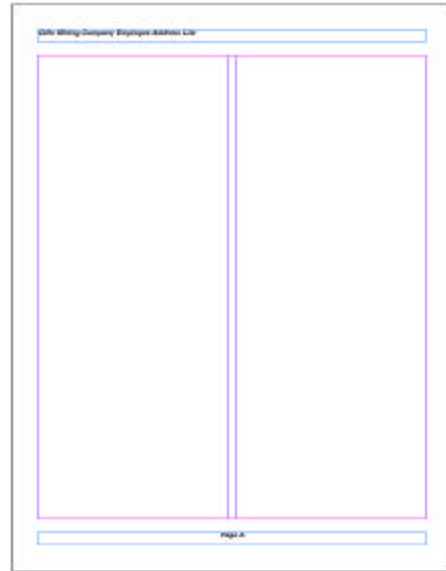
The page header—***Odin Mining Company Employee Address List***—and footer (not shown in the previous illustration) are created on the document's master page. Turn to the document's master page now.

- 3 Select the **Window=>Pages** menu path to open the **Pages** window (if necessary). Then, double click on the icon for Master Page A.

InDesign master pages are used to set up the default format of normal document pages. They may also be used to place standard text and graphics onto every page of the document. For example, the header and footer in Tutor\_1 were created by making new text frames on the master page, outside of the main text thread, and then typing text into them. These text frames and their contents will be copied automatically to every page of the document as it is created.

The master page for this document looks like this:

The document's headers and footers are placed in separate text frames on the master page. They will appear automatically on every document page using this master page.



The static text frames on the master page play no role during InData data importing. Rather, the data import process is controlled by some special text known as a **prototype**, which is placed on the first *regular* document page, usually into a text frame which is part of the document's main text thread or into a text frame on the pasteboard of the first spread within the document.

This is a general principle which applies to all InData importing: **master pages** are used to set up the layout of the document pages—size and placement of text frames, number of columns, headers and footers, and so on—while the prototype, which controls data importing, resides in a text frame located somewhere on the *first document page* or the pasteboard beside it.

- 4 Go back to page 1 by double clicking its icon in the **Pages** window.

## Examine the InData Prototype

Here is the InData prototype found in the large text frame on page 1 of Tutor\_1:

```
«fields lastname, firstname, address, city, state, zip, phone»
«firstname» «lastname»↵
«address»↵
«city», «state» «zip»↵
«phone»¶
#
```

The first thing to notice about the prototype are all the left and right chevrons (European-style quotation marks): « and ». Chevrons are used to distinguish the InData prototype statements and placeholders from other, literal text within the prototype.

The first line of this prototype is used to specify the structure of the data file to InData. It begins with the keyword **fields**, which is followed by a list of field names, separated by commas. These names are used to identify the corresponding field in the remainder of the prototype. The names may be completely arbitrary and need not correspond to the actual field names in a database or spreadsheet application, though a prototype is easier to deal with if they correspond exactly. If necessary, the **fields** statement can continue onto more than one line.

The **fields** statement in our prototype indicates that the records in our data file each have seven fields, which we are naming **lastname**, **firstname**, **address**, **city**, **state**, **zip**, and **phone**; the fields in each data record appear in the same order as in the **fields** statement:

«fields	lastname,	firstname,	address,	city,	state,	zip,	phone ¶
	Crashaw	Richard	928 St. Teresa Terrace	Iconia	NM	72637	373-291-2771
	Greville	Fulke	876 Caelica Lane	Loreda	TX	56293	747-828-2837
	Howard	Henry	12435 Surrey Rd.	Earlin	CA	98773	767-293-8372
	Jonson	Benjamin	8211 Fox Dr.	Valpone	SD	72622	

The remaining lines of the prototype specify how the various fields of each record should be arranged in the document and the format that each field should have. In general, the form «*fieldname*» in a prototype is called a **field placeholder**. A field placeholder tells InData to immediately insert that field's contents as each record is imported and to format it the same way that the field placeholder is formatted. Keep in mind that the name of the field is as defined in the prototype's **fields** statement; it is not necessarily the name of the field in its originating database or spreadsheet application.

For example, the field placeholder «**lastname**» in the second line of our prototype directs InData to place the contents of the **lastname** field—which is the first field in each record, since **lastname** appears first in the **fields** statement—into the field placeholder's position as each record is imported, and to format it the same way that the characters «lastname» are formatted: in this case, in 12 point boldface Helvetica type.

Any characters within the prototype not enclosed in « and » marks are carried over literally into the formatted output. Thus, our prototype inserts a comma between the city and state fields, and also places spaces and line breaks at various points between fields.

The complete prototype specifies that four lines of text will be inserted for each record in the data file. All of the lines will be in 12 point Helvetica type, and the first line of each imported record will be in boldface.

InData follows the prototype's directions as it imports each record from the data file. After it finishes formatting each record, it returns to the beginning of the prototype and then begins processing the next record from the data file.

### Look at the Prototype's Paragraph Attributes

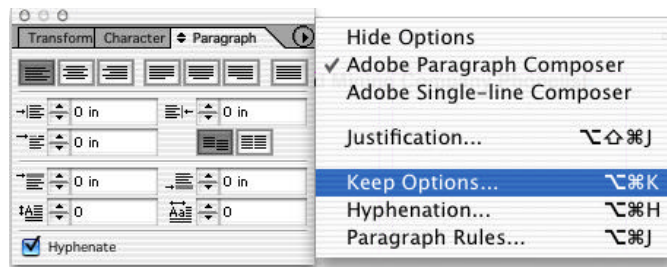
- 5 If the carriage returns and the end of paragraph marks in the prototype are not visible on your screen, select **Type⇒Show Hidden Characters**.

Notice that the prototype consists of two paragraphs. The **fields** line is one paragraph, and the remaining lines form the second paragraph. The arrow at the end of the third, fourth, and fifth lines is a new line character, entered by pressing Shift-Return. It tells InDesign to start a new line at that point, without beginning a new paragraph.

- 6 Place the insertion point on the third line of the prototype and then select the **Paragraph...** option from the **Type** menu. The **Paragraph** settings palette will appear.

Imported records will take on paragraph settings as well as character formats from their field placeholders. In our prototype, notice that the **Space After** field has been set to 1 pica.

Examine the **Keep Options** for the paragraph by clicking on the triangle to the left of the panel tabs, and selecting that item.



Notice that the **Keep Lines Together** box is checked, as is the **All Lines in ¶** radio button underneath it. The first setting will place 1 pica of space between entries in the address list, while the latter ones will ensure that no entry is broken across a page or column boundary (since it's all one paragraph).

- 7 Close the **Keep Options** window without changing any settings. Save the file to a convenient location (use any name you choose).

The character and paragraph formats of each portion of the prototype will be carried over into the formatted data records. If prototype characters or paragraphs have a style applied to them, the imported records will too.

## Import Data into the Document

Now we are ready to import our data. There are two steps required to import data with InData:

- ◆ Telling InData where the prototype is, and placing the cursor at the place in the document where the imported records should go. In the simplest case, both of these may be accomplished simply by placing the cursor anywhere within the prototype (without selecting any text).
- ◆ Selecting an import source from the **InData** menu.

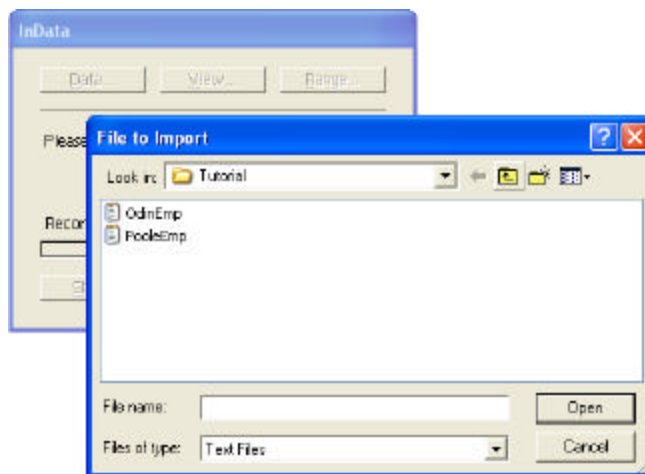
Let's try it:

- 8 With the text (content) tool selected, place the cursor anywhere within the prototype, making sure that you aren't actually selecting text (i.e., you should see a blinking cursor, not a selection region).
- 9 Select **InData => Import from File...** from the InDesign menu. The InData control panel will appear on the screen followed instantly by the InData file specification dialog.

We will use this dialog to specify which data file to import. When we refer to the data file containing records for import into InDesign using InData, we're not talking about the original database or spreadsheet file where the database is entered and modified. InData cannot read native database or spreadsheet file formats. Rather, you must export a text version of the data to a separate "data snapshot" file before importing it with InData. This step has already been performed for you in these tutorials. Exporting data is discussed in detail in Chapter 5.

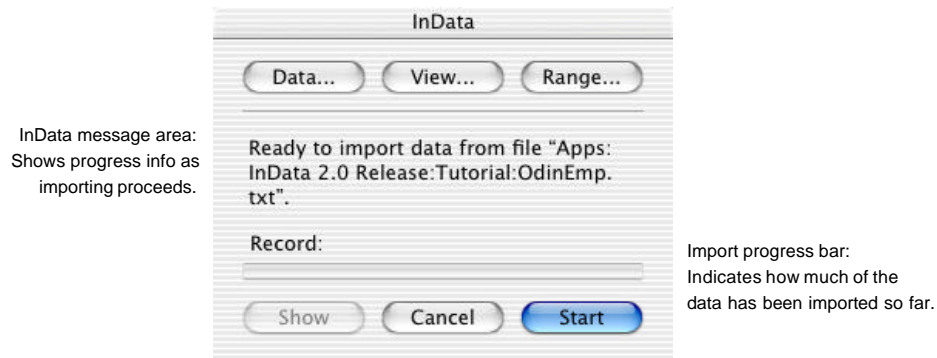
InData control panel  
(in back)

InData import file  
selection dialog



- 10 Select the file named **OdinEmp.TXT** from the list and click **Open**. Once a file has been selected, the InData control panel will be completely visible.

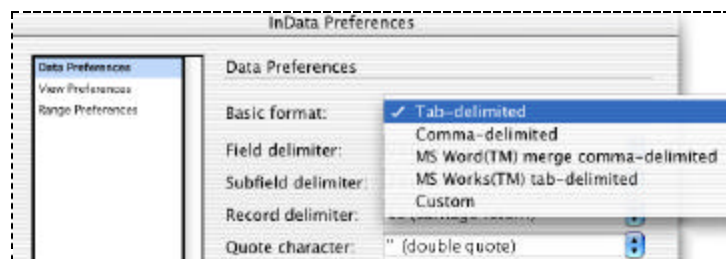
The buttons on the top of the InData control panel bring up dialogs which may be used to set the characteristics of the import process before it begins. They specify the format of the data file (**Data...**), which records to import (**Range...**), and how often to update the screen while importing and formatting data from the data file (**View...**).



- 11 Click the **Data...** button to go to the **Data Preferences** dialog.

The **Data Preferences** dialog contains a plethora of settings and can seem daunting at first. However, for most import operations the default settings work just fine. We'll only be looking at the **Basic format** pop-up menu. This menu specifies how fields within records are separated in the data file. The two most common methods are to place commas between fields (called *comma-delimited format*) and to place tabs between them (*tab-delimited format*). Our data file is set up with tabs separating the fields in each record.

- 12 If the **Basic format** field is not already set to **Tab-delimited**, then change its current setting by selecting **Tab-delimited** from the pop-up menu:



- 13 Once the file format is set to **Tab-delimited**, click **OK** to return to the InData control panel.

By default, InData is set up to import all of the records from the data file and to update the screen only on request. For this first import operation, we'll use these default settings for data range and document view.

- 14 Click the **Start** button to begin data importing.

As importing proceeds, the bar graph in the status area indicates how much of the data file has been imported so far:



InData will also display messages about how the import is proceeding in the top part of the status area.

It's much faster not to update the document on the screen while importing data, so this is InData's default setting. Once importing is complete, the document view will be updated. In this case, the prototype has been replaced by the imported names and addresses from the data file, formatted just as the prototype was. The completed document will look something like this:

<i>Odin Mining Company Employee Address List</i>	
<b>Richard Crashaw</b> 928 St. Teresa Terrace Iconia, NM 72637 373-291-2771	<b>Andrew Marvell</b> 923 Appleton Heights Rd. Windsor, VT 08253 565-283-1218
<b>Fulke Greville</b> 876 Caelica Lane Loredo, TX 56293 747-828-2837	<b>John Milton</b> 76 E. Wander Way Paradise Valley, CA 96342 223-384-3826

## Undoing a Data Import

Sometimes, once a data import operation is complete, it becomes clear that you made a mistake in the prototype, and you'll want to redo the import. In this case, what you need to do is to go back to the version of the document as it existed before the import operation. Assuming that you have saved the document at that point, the best way to do this is to select **File=>Revert** from the InDesign menu. We'll do that now to illustrate this process.

- 15 Select **Revert** from the **File** menu. Click **OK** in the confirmation dialog that appears.

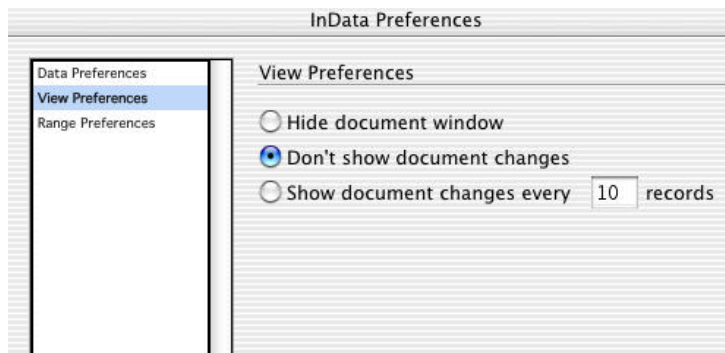
The original prototype reappears in the document window, ready once again to import data.

InData also lets you place the prototype in a separate text frame within the document, rather than in the target text frame. In this case, you can re-do an import operation simply by reimporting over the unwanted data. This option is discussed in the section “Alternate Prototype Placement” in Chapter 4.

## View During Import Options

We'll import the same data into this document once more to illustrate the workings of the options in the **View Preferences** dialog.

- 16 Select **Import from File...** from the **InData** menu. (If this choice is dimmed, it means that you have forgotten to place the cursor in the prototype first.) Select **OdinEmp.TXT** once again from the **Data File** dialog, and then click the **View...** button on the InData control panel.



There are 3 document view update choices during the import operation:

Show no updates at all during import.

Don't update until import is complete.

Update window after each batch of the specified number of records.

- 17 For this import, set InData to update the screen every ten records by clicking the lowest radio button and entering **10** into the frame. Then click **OK** to return to the InData control panel.
- 18 Go to the **Data Preferences** dialog and once again make sure that the **Basic format** field is set to **Tab-delimited**. Click **OK** to return to the InData control panel, and then click **Start** to begin importing records.

This time the screen is updated during data import. Notice how importing stops while the screen is being updated and that the entire process takes longer in this mode.

You have now completed the first InData tutorial. You may go on to the second tutorial below immediately or come back to it sometime later. When you decide to stop, select **Quit** from the **File** menu to end this InDesign session; you need not save the document you just created.

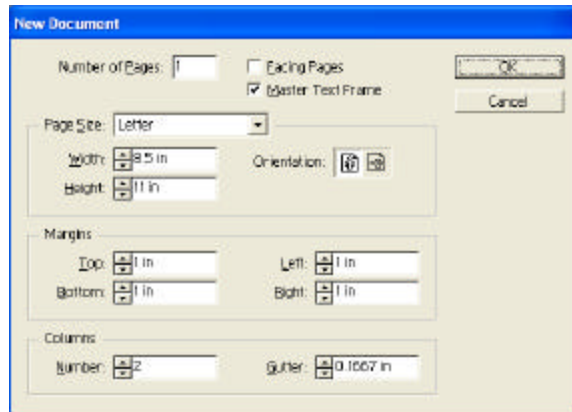


## Tutorial 2: Creating a Company Phone List

In this tutorial, you will set up a complete InDesign document that will become a company phone list. You will place items on the master pages, enter an InData prototype, and use it to import and format the actual data records. Later tutorials will build on the relatively simple document you create here.

### Setting up the Document

- 1 Create a new InDesign document by selecting **New...** from the **File** menu. Create a one-sided, US Letter document (uncheck **Facing Pages** box), having two columns. Choose reasonable values for the page margins and spacing between the columns, and be sure that **Master Text Frame** is checked. Some suggested values are illustrated in the illustration below:



You may choose different values if you like, but deviating too far from these suggestions may produce unpredictable results. When you click **OK**, InDesign will open the new document and display page one in the document window.

### Placing Fixed Text and Graphics on the Master Page

The first thing we'll set up are our master page items.

- 2 Turn to the document's master page by opening the **Pages** palette if necessary (select **Windows=>Pages**), and then double click on the icon for Master Page A in its top section.

We're going to add a header and a graphic to the master page which will appear on every page of the finished document.

- 3 Select the text frame tool, and place a text frame above the two main text columns on the master page. We placed the top of ours about 1/2" from the page edge. Make the text frame extend from the left to the right margins.

Once you've drawn the frame, enter the following text into it:

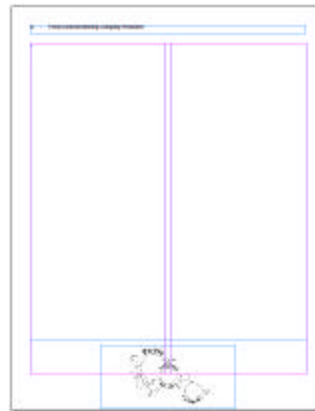
**A        Poole Asteroid Mining Company Phonelist**

The first character—which appears as **A**—is the InDesign page number placeholder. It is produced by choosing **Layout=>Insert Page Number**. Follow this character by a tab, and then the literal text above.

The header you've created will place the page number and document title in the upper left corner of every page. Feel free to format the header text however you'd like (we chose 9-point Helvetica bold). Next, we'll add a graphic to the master page.

- 4    Select the rectangular frame tool and draw a picture frame at the bottom of the page. Make the picture frame about 1.25" x 2".
- 5    Adjust the text frame so that its bottom edge is above the picture box.
- 6    Select **Place** from the **File** menu. Select the file P\_Logo.TIF from the list, and then click **OK**.
- 7    Once the graphic appears, size it for the frame you've drawn by **Object =>Fitting =>Fit Content Proportionately**. This command sizes the picture to fit the frame while maintaining its aspect ratio. If you want, you can also fine tune the picture's placement within the frame by hand.

Your completed master page will look something like this one.



We'll now turn to the job of creating the InData prototype.

### Creating the InData Prototype

The InData prototype does not go on the master page. Rather, it is placed on the first document page where you want the imported records to start. So we'll need to turn back to page one before starting the prototype:

- 8 Turn back to page 1 of the document by double clicking on its icon in the **Pages** palette.

Notice that the header and graphic you put on the master page now appear on page 1. Since this is an actual document page, the page number placeholder has been replaced by the actual page number (1 in this case).

- 9 Select the content tool and place the cursor at the top of the left text column. (You may want to choose **View =>Actual Size** first.) This is where the prototype will go.

For this phone list, we'll be using the first four fields of each record:

- ◆ The last name of the employee
- ◆ The employee's first name
- ◆ The employee's phone extension
- ◆ The employee's title.

Here are some records from the data file:

Calvin	John	2873	VP Personnel	...
Castiglione	Baldasar	2283	Sales Rep	...
Donne	John	2847	Senior Trainer	...
Drake	Francis	2871	Development Engineer	...

We'll call these first four fields **last**, **first**, **ext**, and **title**, respectively. You're now ready to type in a **fields** definition statement for this file.

- 10 Type in the following **fields** statement, ending with a carriage return:

**«fields last, first, ext, title¶**

Chevron characters are typed using the following keystrokes:

	OPENING CHEVRON: «	CLOSING CHEVRON: »
Macintosh	Option-\	Shift-Option-\
Windows	Alt-0171	Alt-0187

You can also use the **Type=>Insert Special Character** menu, available by right clicking under Windows or control clicking on the Macintosh. Note that all InData special characters have been added to the menu.

Notice that the last character of the prototype statement line is a paragraph marker, produced when you entered the carriage return (select **Type=>Show Hidden Characters** if paragraph marks in the text are not already visible). When an InData prototype statement ends with a carriage return, and the return isn't part of the desired result, then the right chevron is not needed.

Next, you will specify how the data is to be entered and formatted. The final entries will look like this:

**Boccaccio**, Giovanni ..... Ext. 2176  
*Secretary*

- 11 Type in the following prototype lines:

```
«last», «first» Ext. «ext»  

«title»
```

The right chevron in reversed colors indicates a tab; it will appear in light blue on the screen. The character at the end of the first line is a new line (Shift-Return), and the prototype ends with a carriage return, producing the end of paragraph mark. We will set up an appropriate tab stop to separate the extension from the name a bit later.

The field placeholders indicate that the contents of that field are to be inserted; all of the other characters you typed will be inserted literally into each record as it is formatted. Thus, this prototype will enter the contents of the **last** field first (it also happens to be the first field in the data file), followed by a comma and a space. Then it will insert the **first** field, followed by a tab and “Ext.” Then it will insert the third field in each incoming record—**ext**—followed by a line break (but not starting a new paragraph). Finally, on a second line for each record, InData will insert the contents of the **title** field.

The prototype ends with a carriage return, starting a new paragraph. Thus, each record in the data file will become its own paragraph in the finished document. In this case, the paragraph mark is a literal character within the prototype, since we want InData to start a new paragraph for each imported record. Thus, the right chevron is included on the **title** field placeholder, so the ending paragraph return is treated as literal text instead of being “consumed” as the end of an InData prototype statement.

Now you will specify the formatting for the prototype. First of all, the last name should be in boldface:

- 12 Select the *entire* field placeholder for the **last** field, *including* the « and » marks. Make those characters bold (using the **Character** palette).
- 13 Using the same method, select and format the **title** field placeholder to change its formatting to italic type. You may also make any other formatting changes that you want to.

The final step in preparing the prototype is to set the tab stop before the extension field. We’re placing our tab stop at the right margin of the text column.

- 14 Place the cursor in the second line of the prototype and then select **Tabs...** from the **Type** menu. Set a right tab at at the right margin by clicking on the right tab icon. Then, click in the ruler above the text frame near the right margin, and adjust the new tab stop if necessary. Return to the **Tabs** palette and set the tab leader character to a period, and then close the palette.

Finally, set the space after setting for this paragraph to 12 points (0.1667").

The fully formatted prototype appears below. It is now ready to import data.

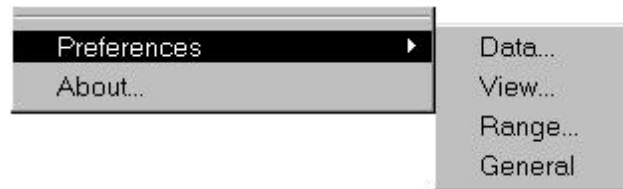
```
«fields last, first, ext, title¶
«last», «first» » ..... Ext. «ext» ¬
«title»¶
```

## Setting InData Preferences

The last time we imported data with InData, we had to specify the data file format and screen updating selection each time. In this tutorial, we'll make the appropriate settings permanent for this document.

This is accomplished by setting InData preferences, accessed via the **Preferences** item on the **InData** menu. As is true of InDesign preferences, whenever preferences are set while a document is open, the settings apply to that document alone. When they are set without any open document, they apply to every document that has not set its own specific preferences.

Selecting **InData=>Preferences** produces a slideoff menu:



The **Data...** selection brings up the **Data Preferences** dialog. Changes that you make to InData's default settings will take effect for every subsequent data import operation in the currently open document.

- 15 Set the default data format preferences for this document to tab-delimited by selecting **Preferences=>Data...** and then choosing **Tab-delimited** from the **Basic format** pop-up menu. Click **OK** to exit from the dialog.

The **View...** item on the InData **Preferences** menu allows you to specify the document's **View Preferences** defaults. Set this default now if you wish.

The **Range...** item allows you to set the range of records which should be imported (the default setting is all records in the data file).

The **General...** item contains a variety of settings, including many specific to picture importing, which we won't consider here. It also allows you to instruct InData to automatically begin importing data once the data file is chosen (rather than waiting for you to press the **Start** button). You may set this option if you like.

Before importing data into a document, it's a good idea to save the document in case you want to use it more than once. If it's something you might want to use frequently, you can save it as a template. We'll do that for our document now.

- 16 Select **Save As...** from the **File** menu to save your work. Select the **Template** button, and enter a filename into the frame (perhaps Tutor\_2). Then click **OK** to exit from this dialog.

## Importing Data into the Document

Since we've set the data file type already, we're ready to import data.

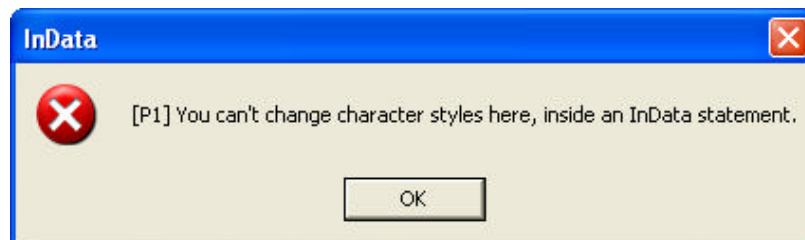
- 17 Place the cursor anywhere within the prototype. Then choose **InData=>Import from File....** When you are prompted with a file selection dialog, choose the file named PooleEmp.TXT located in the InData Tutorial folder. Then press the **Start** button on the InData control panel (unless you set autostart under **General Preferences**, in which case InData starts importing immediately).

When importing and formatting is finished, all the records in your document will look like the sample shown previously in step 10.

## Making a Deliberate Mistake

To conclude this tutorial, we'll expose you to a common error made by beginning InData users.

- 18 Select **Revert to Saved** from the **File** menu to undo the import operation. The, select the right chevron for the **last** field and change the formatting to plain type (not bold).
- 19 Repeat the import operation. You will receive an error message like the following:



As this message indicates, it is illegal to change format styles within a prototype statement, *including the opening and closing chevrons*.

### On Your Own

This completes the second InData tutorial. If you like, you can **Revert to Saved** and experiment further with the prototype and data importing process. In particular, you may want to add some additional fields to the prototype. The complete records in the data file contain the following fields:

- ◆ last name
- ◆ first name
- ◆ phone extension
- ◆ title
- ◆ department
- ◆ home address
- ◆ city
- ◆ home phone
- ◆ picture filename

Ignore the last field for the moment; we'll cover picture importing in Tutorial 5. Try adding the remaining fields to the prototype and reimporting data. The file Completed Tutor\_2 in the Tutorial folder presents one example of a more complicated prototype for this data.

## Tutorial 3: Advanced Data Importing

This tutorial will build on a slightly enhanced version of the template you created in the last tutorial. In it, you will learn to perform the following operations:

- ◆ Adding additional (non-prototype) text to regular document pages. This will enable you to understand the prototype in the context of an entire document.
- ◆ Removing extra blank lines via conditional importing.
- ◆ Constructing new text out of the imported data.

- 1 Open the file Tutor\_3 in the Tutorial folder.

This file is set up much like the file we created for Tutorial 2. It has been modified to use facing pages appropriate for double-sided printing. The header on the master page has been constructed so that the page number is always at the outside edge of the page, and the picture frames for the Poole logo now appear in the outside bottom corner of each page.

- 2 Turn to the master page for this document and examine the header text frames, noticing how the page number and header have been positioned in each one. Note also where the picture frames have been placed on each page.

- 3 Return to page one of the document and examine the prototype in the main text frame. It looks like this:

```
«fields last, first, ext, title, dept, address, city, phone»
«last», «first» . . . . . Ext. «ext»
«title» («dept» Dept.)
«address», «city»
«phone»
```

The **fields** statement is longer and describes four more fields than the previous one. Two additional lines are included in each formatted record, containing the employee's home address and phone number, set in a smaller type size than the rest of the entry. A formatted record will look something like this:

```
Shakespeare, William . . . . . Ext. 1122
Head Writer (Marketing Dept.)
1021 Arden Way, Avon
555-1212
```

We'll now begin making changes to this document.

### Adding a Title on the First Page

The first modification we'll make is to add a title on page one of the document. We'll do this by creating a text frame on page one outside of the main text thread, which goes through the large frame on each page.

- 4 Resize the text frame on page 1 so that its top edge is about 1 inch below the top margin (i.e., at the 2" or 12 pica position on the ruler).
- 5 Create a new text frame just under the top margin, extending from the left to the right margin. Make the frame about 1/2" high.
- 6 Enter the following text into the text frame:

**Poole Asteroid Mining Company Phonelist**

Now we'll format the title text by applying a InDesign style sheet.

- 7 Leave the text tool in the title you've just created. Then choose the **Type=>Paragraph Styles** menu path to open the **Paragraph Styles** window (if necessary) and then click on the **Title** style to apply it to the current paragraph. Notice how the title becomes centered, boldface and larger.

Because the text frame you just created is not part of any text chain and does not appear on the master page, it will appear only on page one of the document, with the exact content and in the exact position as when you set it up.



## Avoiding Unwanted Blank Lines From Missing Fields

If we imported data into the document right now, some of the resulting name and address entries would have unwanted blank lines within them:

**Calvin, John** . . . . . Ext. 2873  
*VP Human Resources* (Admin Dept.)  
 264 Poli Way, Shellville  
 444-9584

**Caxton, Thomas** . . . . . Ext. 2846  
*Accountant* (Admin Dept.)  
 3421 W. 120th St., Bright Valley

*missing phone*

**Donne, Anne** . . . . . Ext. 2190  
*Secretary* (Admin Dept.)  
 3981 W. 115th St., Bright Valley  
 444-5984

Thomas Caxton's record has no data in the phone number field. However, InData still puts a new line character into the document at that point because it is a literal character within the prototype.

InData is able to conditionally import fields into the document. Using this feature, we can eliminate the blank lines by telling InData to import certain fields and their corresponding new line characters only if the fields are not empty. Here is an example of how this is done (but don't type anything into your prototype yet):

```
«if address is not empty»«address»␣
«endif»¶
```

These prototype statements tell InData to insert the contents of the field named **address** into the document followed by a new line character only if that field is not empty.

Sometimes, however, it is the new line character *itself* that needs to be made part of the condition. Here is an example:

```
«lastname», «firstname»«if title»␣
«title»«endif»¶
```

In this case, the **title** field is being conditionally imported. If a record has a title, then InData will insert a new line character after the name line. However, if a record has no value in the **title** field, then this new line character is not needed. That is why the new line is placed *before* the **title** field placeholder but *inside* the **if...endif** conditional statement.

This prototype also illustrates a short version of the **if field is not empty** statement: the **is not empty** part is actually optional.

We'll add conditional importing statements to both the **address** and **phone** fields in our prototype.

- 8 Modify the prototype so that both the **address** and **phone** fields are imported only if they are not empty. Only include the **city** field if the **address** field is not empty. You may want to try this on your own, before looking at the completed prototype below.

This is how the modified prototype will look:

```
«fields last, first, ext, title, dept, address, city, phone»
«last», «first» » . . . . . Ext. «ext» ¬
«title» («dept» Dept.) «if address is not empty» ¬
«address», «city» «endif» «if phone» ¬
«phone» «endif» ¶
```

There are a couple of tricky aspects to this prototype. First, the **if** statements need to place the new line characters *before* their corresponding fields, so that new lines are created only if there is data in the field. Secondly, the final **endif** statement comes *before* the final new paragraph mark because the latter should not be imported conditionally; each record should always end with a paragraph break.

This section has illustrated only the simplest uses of InData's conditional importing features. See Chapter 6 for an in-depth discussion of these capabilities.

## Adding Room Numbers

The last modification we'll make to this file is to add each person's office location to the formatted record. In this company, in-house phone numbers in the **ext** field are created by starting with a 2, then adding the floor of the building the person's office is on, and finally adding the two-digit room number after it. For example, an extension of 2287 means that person's office is on the second floor in room 87. So, to add floor and room numbers to the outputted records, we need to extract this information from the **ext** field.

InData has a **character** operator that allows you to do just that. Here is its general format:

```
«character start to end of field»
```

where *start* and *end* are numbers indicating which characters to extract (numbering begins at 1), and *field* is the name of a field from the fields statement. If you want only one character from the field, then the **to end** part may be omitted. If you want all the characters starting at a given location, then use the form

**length**(*field*) for *end*, which stands for the total length of the field's contents. The keyword **character** may be abbreviated to **char**.

We'll add a line to each imported record below the name and extension line, containing the floor and room number of their office. Here is how a formatted record will look:

**Johnson**, Samuel. . . . . Ext. 2456  
 Floor 4, Room 56  
*Installation Planner* (Customer Support Dept.)  
 123 North Cedar Street, Berkeley  
 555-6677

- 9 Add **char** expressions and literal text to the prototype to produce records like the one above. You may want to try this on your own before looking at the completed prototype.

Here is what the finished prototype looks like:

```
«fields last, first, ext, title, dept, address, city, phone»
«last», «first» » . . . . . Ext. «ext»
Floor «char 2 of ext», Room «char 3 to 4 of ext»
«title» («dept» Dept.) «if address is not empty»
«address», «city» «endif» «if phone»
«phone» «endif»
```

The words “Floor” and “Room” are literal text placed into the prototype. The appropriate characters extracted from the ext field are placed after each word. Don't forget to change the font size of the line you've added (if desired).

As this prototype indicates, it's perfectly all right to use a field more than once in a prototype. Incoming fields can be placed in any order within a prototype, be used more than once, or be ignored altogether. It's completely up to you as the prototype creator.

Don't worry if some of the prototype lines wrap to the next column. The formatted records will fit within the column just fine.

- 10 If you like, save the file as a template for later use.

### Import the Data

Now we're ready to import the data.

- 11 Check to make sure that the data format is set to **Tab-delimited**.

- 12 Select **InData=>Import from File...**, and then select the Poole employees data file once again.

When all of the data is imported, you will have completed tutorial number three. In the next tutorial you will learn some more advanced uses of conditional statements in InData.

## Tutorial 4: Adding Department Headlines

In this tutorial we'll modify the prototype from Tutorial 3 to add reversed type department headlines. The formatted records will look something like this:

Sales
<b>Beaumont</b> , Christopher ..... Ext. 2637 Floor 6, Room 37 <i>Sales Representative</i> 2312 W. 112th St., Bright Valley 444-3322

- 1 To begin this tutorial, either open the file you created in the last tutorial (assuming that you didn't import data into your only copy) or open the file Tutor\_4 in the Tutorial folder. If you choose the latter file, you will notice small formatting changes in the prototype.

### Comparing the Current and Previous Records

The records in the data file are sorted by department and then by last name; the "Admin" (administration) department comes first, with all its members in alphabetical order, followed by the alphabetized members of the Marketing department, and so on. We want to create a new department headline every time the department changes (and only at those times). To do this we'll use the **if** statement.

InData's **if** statement is more powerful than the simple use we made of it in the last tutorial might suggest. For example, it may also be used to compare the contents of a field to a literal value, to the contents of another field, or to the value of the same field in the previous record. It's the latter operation that we want.

- 2 First of all, remove the **dept** field placeholder and its accompanying literal text—"Dept.," and the preceding opening parenthesis—from the prototype.

We're going to place the department headline before the first record for each department, so we'll be adding to the prototype on a line between the **fields** statement and the **last** field placeholder.

- 3 Create a new line for the **dept** placeholder by entering a carriage return at the end of the **fields** statement, and type in the **dept** placeholder on that line. Set the department placeholder in 16 point boldface Helvetica type.

The prototype should now look something like this:

```
«fields last, first, ext, title, dept, address, city, phone»
«dept»
«last», «first» » ..... Ext. «ext»
Floor «char 2 of ext», Room «char 3 to 4 of ext»
«title»«if address is not empty»
«address», «city»«endif»«if phone»
«phone»«endif»
```

As it currently is, the prototype tells InData to place the contents of the **dept** field before every record. Our next step is to tell InData to include the **dept** field only if it has changed since the last record.

- 4 Add this **if** statement and its closing **endif** statement to the prototype so that the dept field is imported only if its contents are different from that in the previous record:

**«if dept is not previous dept»**

This statement compare the contents of the **dept** field in each record with what was present in that same field in the previous record. If the **dept** field changes, then InData will perform the actions that lie between the **if** statement and its closing **endif** statement.

Here is the completed prototype:

```
«fields last, first, ext, title, dept, address, city, phone»
«if dept <> prev dept»
«dept»
«endif»«last», «first» » ..... Ext. «ext»
Floor «char 2 of ext», Room «char 3 to 4 of ext»
«title»«if address is not empty»
«address», «city»«endif»«if phone»
«phone»«endif»
```

As this prototype indicates, the **previous** keyword may be abbreviated as **prev**, and the **is not** operator can be abbreviated as **<>** (the not equal sign).

We've placed the **if** statement on a separate line above the **dept** field placeholder, ending it with a carriage return instead of a right chevron mark so that the carriage return does not become a literal character in the prototype and produce

blank lines in the formatted records. However, we could have written that part of the prototype as:

```
«if dept is not prev dept» «dept»¶
«endif»
```

This is purely an aesthetic choice; either form is correct. Note that the **endif** statement comes after the carriage return following the dept field placeholder since the carriage return itself should only be included if the **dept** field is.

We also chose to set the **if** statement in smaller type than the **dept** placeholder, to minimize its visual prominence as we look at the prototype. Since it won't appear in the formatted data, we are free to format it as desired.

You might wonder if this prototype will work correctly on the first record of the data file. The answer is yes, since the previous contents of any field in the first record are considered empty (null).

### Creating Reversed Type with Paragraph Rules

Now we'll turn to the task of setting the paragraph format for the **dept** field placeholder.

- 5 Place the cursor in the paragraph holding the **dept** field placeholder. Bring up the **Paragraph** dialog by choosing **Type=>Paragraph...** from the dialog's pop-up menu. Change the following settings (you may alter the numeric values if you like, to taste):

<b>Space Before:</b>	<b>0.5" (36 pt)</b>
<b>Space After:</b>	<b>0.2" (14.4 pt)</b>
<b>Keep with Next:</b>	<b>1</b>
<b>Alignment:</b>	<b>Centered</b>

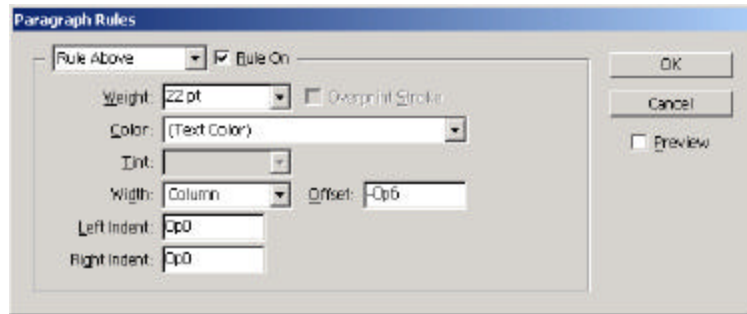
- 6 Select the entire **dept** field placeholder and change its color to white.

The placeholder will temporarily disappear, but it will reappear as soon as we add black rules to the paragraph:

- 7 Without changing the insertion point, select **Paragraph=>Paragraph Rules....** from the **Paragraph** palette's popup menu. Add a 22 point black rule above the paragraph. Set the values of the **Offset:** field so that the rules form a black frame behind the **dept** field placeholder.

The idea here is to get the rule to cover the text completely. Since the text is white, it will stand out against the black rule. We found that an offset of about .083" (-6 pt) works well for 16 point boldface Helvetica type. If you're using a different font or size, you may need to experiment with this value a bit.

Here is our completed **Paragraph Rules** dialog:



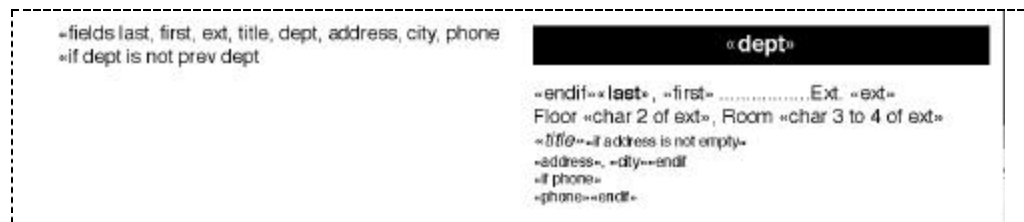
- 8 Save the file at this point as we'll want to use it more than once. If you like, save the file as a template for repeated use.
- 9 Import the Poole employees data file into the document. Several department headlines will be created along with the formatted employee records.

### Forcing a Paragraph to the Top of a Column

Sometimes, you want to force a paragraph to the top of a column or page. In this example, you might want each department to start its own column. Here is how to accomplish this.

- 10 Revert to the last saved version of the document, or open the template you created. (If you didn't save it, you can open the file named Tutor\_5).
- 11 Place the cursor in the paragraph containing the **dept** field placeholder, and then open the **Paragraph** palette. Click on the palette's slide-out menu, and select the **Keep Options...** item. Then, select **In Next Column** from the **Start Paragraph** menu. Close all dialogs.

This will force the department header to the top of the next column on the page:



However, when you import the actual data, the initial fields and **if** statements will not be placed into the document, and the first department headline will start at the top of the first column on page 1 of the document.

- 12 Import the data file into the document again, and then examine the document's pages to verify that the department headlines are placed at the start of new columns.

This completes the fourth tutorial.

## Tutorial 5: Importing Pictures

In this tutorial, we'll conditionally import graphics into the document. We'll finally be using the last field of each data record, which contains the name of the file containing each person's photograph (if one is available).

*Note:* to save space, only one photograph file has been placed on the InData distribution, and this same file is used for three different people in the data file. Despite these unrealistic aspects, however, you will still be able to get a good sense of how picture importing works from this tutorial.

- 1 Open the file named Tutor\_5 in the Tutorial folder. Modify the space before setting of the department paragraph as in the previous tutorial. Save the file to any name you choose within the same folder before proceeding, or pictures will not import properly.

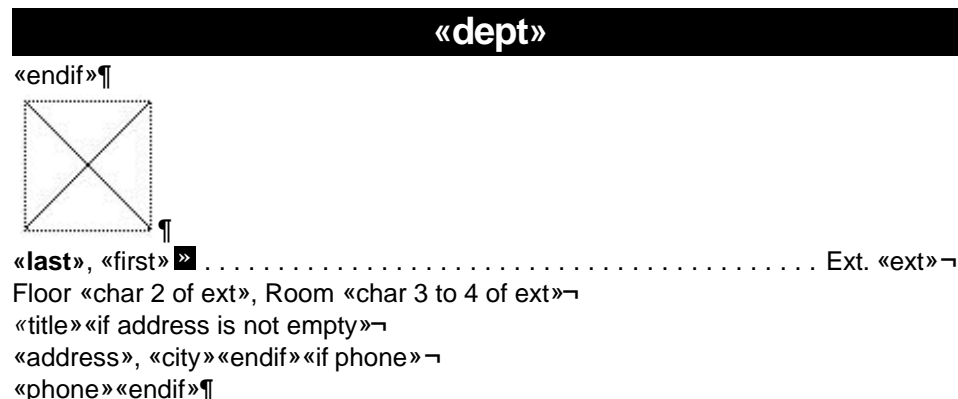
### Adding a Picture Frame to the Prototype

The first step required in importing pictures with InData is to add a picture frame to the prototype.

- 2 With the rectangular frame tool, draw an approximately one inch square picture frame in some convenient location (in the empty right hand column or on the pasteboard, for example).
- 3 With the selection tool, make sure the just-created picture frame is selected, and then choose **Cut** from the **Edit** menu.
- 4 With the type tool, place the cursor between «**endif**» and «**last**» in the prototype. Begin a new paragraph by entering a carriage return.
- 5 Without moving the cursor, paste the cut picture frame into the prototype, and then enter another carriage return.



This part of your prototype will now look like this:



Continue the prototype by modifying the paragraph containing the picture frame, setting the space before to .08" (6 points) and the space after to 0. Set its keep with next setting to 1 line (so the picture always stays with corresponding entry).

### Specifying the Picture Filename in the Prototype

First we need to add the final field in the data file to the prototype's fields statement.

- 6 Add the field name **pix** to the end of the **fields** statement. Be sure to place a comma before it.

The **set filename** command is used to tell InData what field contains the picture filename for any anchored picture frames within the prototype. Its general format is:

**«set filename of picture *n* to *field*»**

where *n* indicates which picture frame within the prototype is meant, and *field* indicates which data field contains the name of the picture file to import. We want to set the field for the first (and only) picture frame in the prototype to **pix**.

- 7 Add a **set filename** statement before the **last** field placeholder. You may end the **set filename** statement with a carriage return, placing it on its own line within the prototype, if you omit the right chevron.

Here is how this part of the prototype now looks:



«set filename of picture 1 to pix»  
 «last», «first» » ..... Ext. «ext» ¬

### Making the Picture Import Conditional

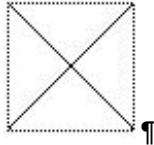
Of course, we only want to include the picture frame if the **pix** field contains a file name—if it's not empty, in other words.

- 8 Add an **if** statement to the prototype so that the picture frame, its associated paragraph mark, and the **set filename** statement are included only if the **pix** field is not empty.

Here is one way to accomplish this:

«dept»

«endif» «if pix»

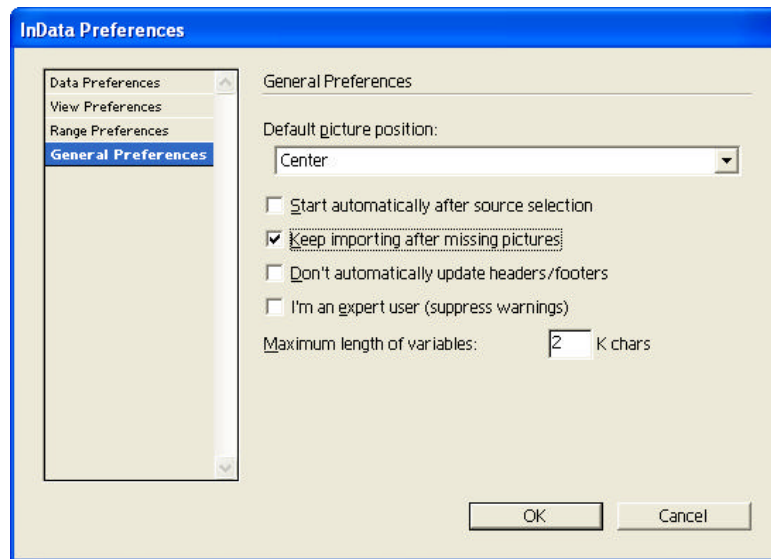


«set filename of picture 1 to pix» «endif»  
 «last», «first» » ..... Ext. «ext» ¬

The **if** statement precedes the picture frame, and the **endif** comes after the **set filename** statement. A right chevron now closes the **set filename** statement, and the paragraph mark closes the **endif** statement.

### Specifying Global Picture Attributes

When you import a graphic into a picture frame, it may or may not be sized correctly for the frame. InData allows you to specify how to position and size imported graphics within their picture frames in the **InData: General Preferences** dialog, which you reach by selecting **InData=>Preferences=>General...** from the InDesign menu.



- 9 Set the **Default picture position** to **Center**, **Size to Fit**, **w/o Distortion**.

### Import the Data

The prototype is now ready to accept data.

- 10 Import the Poole employees data file into the document.

Three of the formatted records on the first page will contain photographs. Here is an example:

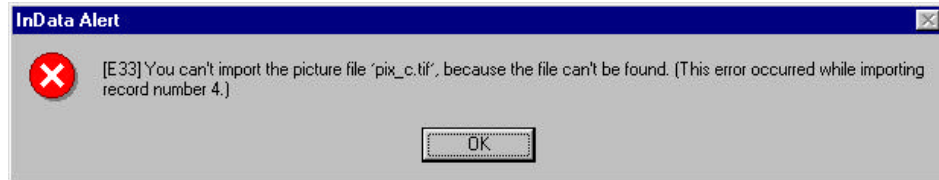


**Pembroke, Mary.** . . . . . Ext. 2839  
 Floor 8, Room 39  
 Accountant  
 3042 W. 118th St., Bright Valley  
 555-9920

(To save space in the InData distribution, the same picture file is used for all three records. For the same reason, the picture file has been compressed in its range of grey, so its quality will not match that of normal scanned photographs.)

### When InData Can't Find a Picture File

When InData can't find a picture file called for during data import, you'll see an error message like this one:



Importing will stop immediately. You can tell InData to ignore missing picture files by turning on **Ignore missing picture files** in the **InData: General Preferences** dialog.

This completes the fifth InData tutorial.

## Tutorial 6: Adding Document Headers and Footers

InData can also produce running headers and footers which change from page to page, based on the contents of the imported data. For example, a telephone directory might have the last names of the first and last people on each page in the top outside corner. In this tutorial, you will learn to create such running headers and footers.

There are three basic steps to creating a running header or footer:

- ◆ Designating one or more fields (or an expression involving them) as the source of the running header/footer text. This involves placing a **mark** or **marked text** in the prototype.
- ◆ Adding a **mark reference** to the header or footer text frame on the document's master page, which serves as a placeholder for the ultimate related imported data or expression, and specifying its characteristics.
- ◆ Importing the data in the usual way. InData creates the running headers and footers after all records have been imported and formatted.

Although running headers and footers will be created automatically when the data is imported, they are not updated automatically if you make subsequent changes to the formatted records. However, you can ask InData to update them at any time. We'll discuss this process in Chapter 9.

To begin this tutorial, we'll set some global InData preferences.

- 1 Before opening any tutorial file, select **InData=>Preferences=>Data...** from the InDesign menu. Set the data format to **Tab-delimited**.

Since no document is currently open, we have just set the default data format for all import operations. InData's preferences work the same way as InDesign's own preferences do in this respect. You may still set an individual document's preferences by setting InData preferences while that document is open, and you may always override default settings by explicitly changing them with the InData control panel before starting an import operation.

- 2 Set InData's global default document view during update behavior by selecting **InData=>Preferences=>View...** from the InDesign menu and then specifying the mode you prefer using.

We're now ready to open the file for this tutorial.

### Creating a Mark in the Prototype

- 3 Open the file named Tutor\_6 in the Tutorial folder. Save the file in the same folder to any name you choose.

We're going to modify this document's header so that it includes the name of the first department listed on each page.

First, we'll modify the department field placeholder so that we can use it in the running header.

- 4 Edit the **dept** field placeholder so that it reads as follows (we've omitted the reversed type and rule):

**«put dept marked "D"»**

This statement tells InData to insert the contents of the dept field as this point in each imported record—just as **«dept»** did before—and to give it the name **D**. The form **«field»** is short for **«put field»** in general, and the **marked** keyword assigns a name to the **dept** field placeholder which can be used to create running headers and footers.

You may wonder why we have to assign a new name to the **dept** field when it already has one. The **marked** keyword is actually quite flexible, and it may be used to mark not only field names but any expression; **char 1 of lastname**, for example. In the latter case, a name needs to be assigned to the expression in order to refer to it in the running headers or footers.

Note that all InData mark names consist of a single letter whose case is ignored (**d** is the same as **D**).

## Adding a Mark Reference to the Master Pages

- 5 Turn to the document's master pages and position them so that you can edit the left header text frame. You may need to change the document view to do so.

Currently the header looks like this:

**D      Poole Asteroid Mining Company Phonelist**

It contains the page number, followed by a tab, followed by the text "Poole Asteroid Mining Company Phonelist." We're going to replace the latter text with the name of the first department on each page.

- 6 Replace the text **Poole Asteroid Mining Company** with **Dept.**

This text will become a mark reference for mark **D**, and thus serve as a placeholder for the contents of the first department field on each page. In the next step, we'll link this text to the mark we set up previously. First, however, we'll add some placeholder text to the right master page header.

- 7 Move to the header text frame on the right master page, and replace the text **Poole Asteroid Mining Company** with **Dept** also.

As what we've done indicates, the text used as mark references in running headers is simply any literal text that you choose to use. It requires no special chevron marks. It's a good idea to choose text that will remind you of what will replace it when data is imported. Thus, we chose "Dept" for our headers. These mark references in running headers and footers also should be formatted as you want them to look in the final document.

## Informing InData About the Mark Reference

The final step in setting up the running headers is to let InData know about the mark reference text you've created and to associate it with a mark in the prototype.

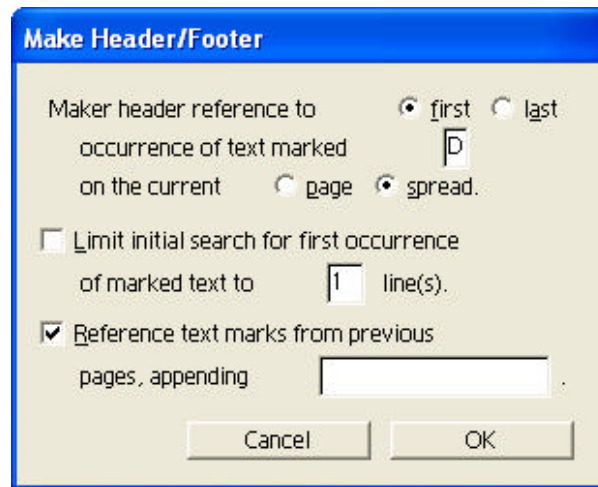
- 8 Move back to the left master page header text frame, and highlight the text **Dept** with the cursor.
- 9 Select **Make Header/Footer** from the **InData** menu.

The **Make Header/Footer** dialog will appear. This dialog allows you to associate the first or last occurrence of a specified mark with the text you have highlighted, transforming the latter into a mark reference.

Selects which occurrence of the specified marker is to be used.

Specifies how far down the page (# lines) to look for the "first" marker.

Marked text can be carried over from a previous page when no marked text is present, with optional added text (e.g. "continued").



Choosing **first** means that the first instance of the marked field or expression on each page (or spread if **spread** is checked) will be placed in the placeholder's location on the page. Similarly, turning on the **last** radio button means that the final instance of that marked field or expression on each page (or spread) will go into the page's header (or footer).

Since we are currently on the left master page, we want the first instance of the department field to go into the header; on right document pages, we'll want the last instance. Here is how the headers on a completed spread might look after data importing:

## 2 Admin

## Marketing 3

First, we'll designate the text we selected before opening the **Make Header/Footer** dialog to refer to the first instance of the department field on the page:

- 10 Change the settings in the dialog to associate the placeholder with the first occurrence of mark **D**—the **dept** field in this case—on the **page**.
- 11 Check **Reference marks from previous pages**, so that if no **dept** field in-line headers appear on a document page, the most recent one from a previous page will be used. We'll just use the department name as is, without appending anything to it.
- 12 Click **OK** to leave this dialog.

InData now views the text as a mark reference.

- 13 Move to the right master page, and select **Dept** in its header text frame.

- 14 Select **InData=>Make Header/Footer**.
- 15 Adjust the settings to select the last occurrence of the marker **D** on the spread. Check **Reference marks from previous pages**. Then click **OK** to exit from the dialog.

You've now associated the placeholder you created on the right master page with mark **D**. The document is now ready for data importing.

### Save the Document as a Template

Before importing data, save the document as a template, by selecting **Save As...** from the **File** menu, entering a filename into the space, and selecting the **InDesign Template** menu option. Then click **OK** to complete the save.

This step is a good idea in general for two reasons. First, it allows you to correct the prototype and try a data import operation a second time if there are problems the first time. Second, it enables you to create new, additional documents using these same formats at a later date.

### Import the Data

- 16 Return to page 1 of the document, and place the cursor anywhere within the prototype. Then import the Poole employees file into the document.

Once data importing is complete, InData will begin to work on the headers and footers, and a second status dialog will appear to indicate its progress:



- 17 Examine the header on each page (after page 1) of the completed document, noting how the text **Dept** has been replaced by the departmental affiliation of the first or last person on that page.

If your document's headers are not correct, you can examine the file **Completed Tutor\_6** in the **Tutorial** folder, which has the completed prototype and the header mark references correctly installed.



## Tutorial 7: Automated Document Creation

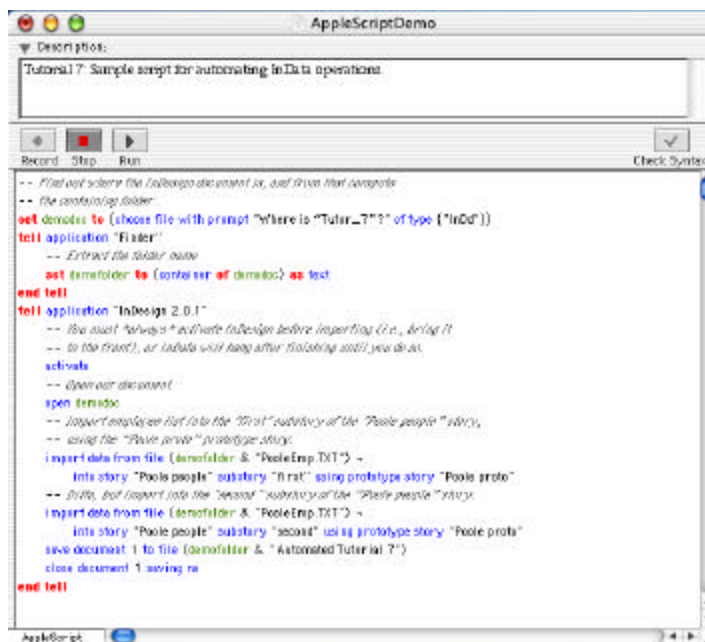
This tutorial introduces automated document building with InData and the native automation facilities provided by the Windows and Macintosh environments. The first subsection of this tutorial appears in two versions; Windows users should skip ahead to the Windows version.

### Macintosh Systems: AppleScript

On Macintosh systems, you can automate data importing with InData and create the resulting InDesign documents via the AppleEvent facility. This tutorial will introduce you to these InData capabilities. It assumes that you are familiar with the basics of AppleScript and that this facility is active on your Macintosh (see Chapter 12 for details).

In this tutorial, we will first demonstrate how the importing process works with AppleScript and then guide you through the creation of a scriptable InDesign document.

- 1M** Make sure that InDesign is running. Close all open documents.
- 2M** Double click on the AppleScriptDemo file in the Tutorial folder. This file is a Script Editor file, and that application will be started automatically:



The script appears in the lower part of the window. The buttons just above the script area control execution of the script.

- 3M** Click on the **Run** button, and then select the file Tutor\_7 in the resulting file open dialog (the file is located in the InData Tutorial folder).

Once you have selected the file, InDesign will become active and two data import operations will occur automatically. When they have completed, the resulting file will be saved as Automated Tutorial 7, and the InDesign file will be closed. Finally, control will return to the AppleScript Editor application.

- 4M** Exit from the AppleScript Editor and return to InDesign. Open and examine the Automated Tutorial 7 file and verify that records have been imported and formatted.

Macintosh users should skip the next subsection.

### Windows Systems: VBScript

On Windows systems, you can automate data importing with InData and create the resulting InDesign documents via Windows Automation. You can use any supported language: Visual Basic, VBScript (which we use here as an example), JScript, and so on. This tutorial will introduce you to these InData capabilities. (see Chapter 12 for more details).

In this tutorial, we will first demonstrate how the importing process works with VBScript and then guide you through the creation of a scriptable InDesign document.

- 1W** If InDesign is running, close all open documents.
- 2W** Start Notepad (or any Windows editor), and open the Tutor\_7.vbs file in the Tutorial folder. This file is a VBScript file, and it contains the following text:

```
'
' InData Tutorial 7 Companion Script
'
Option Explicit
Dim fso, dataFolder, myInDesign, myRecordCount, idDontDisplayAlerts
' From InDesign's type library:
idDontDisplayAlerts = 1699640946
Set fso = CreateObject("Scripting.FileSystemObject")

dataFolder = fso.GetFile(WScript.ScriptFullName).ParentFolder.Path & "\"
Set myInDesign = CreateObject("InDesign.Application")
' You must *always* activate InDesign before importing
' (i.e., bring it to the front), or InData will hang after
' finishing until you do so.
myInDesign.Activate

' Open our demonstration document
myInDesign.UserInteractionLevel = idDontDisplayAlerts
myInDesign.Open (dataFolder & "Tutor_7.indt")
```

```
' Import the employee list into the "first" substory of the
' "Poole people" story, using the "Poole proto" prototype story.
myRecordCount = myInDesign.InDataImportFromFile(dataFolder & "PooleEmp.txt",
"Poole proto", "Poole people", "first")
' Ditto, but import into the "second" substory of the "Poole people" story.
myRecordCount = myInDesign.InDataImportFromFile(dataFolder & "PooleEmp.txt",
"Poole proto", "Poole people", "second")
myInDesign.ActiveDocument.SaveAs (dataFolder & "Automated Tutorial 7.indd")
myInDesign.ActiveDocument.Close
```

After some initial setup activities, the script determines the current folder location and then starts InDesign. Next, it opens the document we will use.

In the main part of this script, the two **myInDesign.InDataImportFromFile** statements (method calls) are where InData is used to actually import the data, using the specified prototype story, into the specified story and substory.

Finally, the resulting document is saved and then closed.

- 3W** Exit from Notepad (discard any changes). Navigate to the InData Tutorial folder and double click on the script's icon. The import operation will begin automatically.
- 4W** Return to InDesign. Open and examine the Automated Tutorial 7 file and verify that records have been imported and formatted.

### Preparing a Document for Automated Data Importing (Windows and Macintosh)

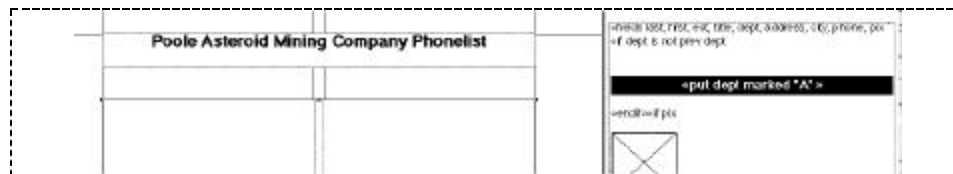
Next, we will modify our Poole employee address list document for AppleEvent or Visual Basic scripting.

- 5** Open the InDesign template you created in Tutorial 6 (or the file CompletedTutor\_6 in the Tutorial folder).

The first step is to move the prototype from the target text frame for the imported data records to a separate text frame on the pasteboard. Placing the prototype in the target text frame is just one of the options for InData prototype placement (this topic is discussed in detail in Chapter 4).

- 6** Select the entire prototype on page 1 of the document and cut it to the clipboard. Then, create a new text frame on the pasteboard to the right of page 1 and paste the prototype into it. Finally, save the document.

The document will now look something like this:



The prototype is located in a text frame on the pasteboard, and the target text frame on the document page is empty.

Next, we will verify that the document is still functioning properly by importing a few data records.

- 7 Click anywhere within the text frame containing the prototype, and then select **InData=>Use Story as Prototype** from the menu. Doing so will cause a check mark to appear next to that menu item.
- 8 Click in the text frame on page 1 of the document, and then select **InData=>Import from File....** Select the Poole employees data file from the file list, and then click OK. If desired, use the InData control panel's **Range...** button to specify just the first few records of the data file. Modify any other settings from the control panel as appropriate. When you are ready to begin importing, click the **Start** button.

The records should be imported and formatted as happened previously. Note that the prototype remains unchanged in its text frame on the pasteboard when importing is done in this way.

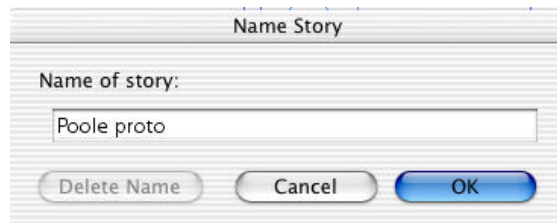
- 9 Select **File=>Revert** to restore the saved version of the document.

The same process is used for automated data importing. We will now continue to set up our document for data importing via AppleScript or Visual Basic.

## Naming Stories

In order to refer to specific text frames in a InDesign document within a script, we must assign labels to them. To do so, we assign a *name* to each *story* (discrete text thread) to which we will want to refer.

A name may be assigned to the current story—the story currently containing the cursor—with the **InData=>Name Story...** menu item, which brings up the following simple dialog:



You simply type the desired name into the text field and then click OK.

- 10 Place the cursor into the text frame containing the prototype. Then select **InData=>Name Story...**, and assign the story the name **Poole proto**.
- 11 Using the same technique, assign the name **Poole people** to the main text thread on document page 1.

Named stories may be similarly divided into **substories**: named subsections of a named story. We will define two substories of the **Poole people** story.

- 12 Make sure that the cursor is still with the text frame containing the **Poole people** story. Then select **InData=>Name Substory...** from the menu, and assign the name **first** to the substory. The substory name will appear in the text frame.
- 13 Create a second substory, named **second**, in a similar manner. Save the document as a template named My Tutorial 7.

### Macintosh: Prepare the AppleScript

Next, we will prepare the script which will control the automated importing process. (Windows users should skip ahead to the next subsection.)

- 14M Open the AppleScriptDemo file in the Tutorial folder by double clicking on it or opening it from within the AppleScript Editor application.
- 15M Immediately select **File=>Save As...** from the menu and give the new file the name My Script.

The existing script will require only minor changes for use with your document:

```
-- Find out where the InDesign document is, and from that compute
-- the containing folder.
set demodoc to (choose file with prompt "Where is \"Tutor 7\"?" of type {"InDd"})
tell application "Finder"
    -- Extract the folder name
    set demofolder to (container of demodoc) as text
end tell
tell application "InDesign 2.0.1"
```

```

-- You must *always* activate InDesign before importing (i.e., bring it
-- to the front), or InData will hang after finishing until you do so.
activate
-- Open our document.
open demodoc
-- Import employee list into the "first" substory of the "Poole people" story,
-- using the "Poole proto" prototype story.
import data from file (demofolder & "PooleEmp.TXT") ~
    into story "Poole people" substory "first" using prototype story "Poole proto"
-- Ditto, but import into the "second" substory of the "Poole people" story.
import data from file (demofolder & "PooleEmp.TXT") ~
    into story "Poole people" substory "second" using prototype story "Poole proto"
save document 1 to file (demofolder & "Automated Tutorial 7")
close document 1 saving no
end tell

```

The script begins by asking the user where the InDesign document is and then determining its folder location.

The main work of importing happens within the **tell application "InDesign" ... end tell** portion of the script. First, the InDesign application is activated, then the document file is opened. The two **import data** statements are where InData is used to actually import the data, using the specified prototype story, into the specified story and substory. Finally, the resulting document is saved and then closed.

The underlined parts of the script will need to be modified for use with your document.

- 16M** Replace the first underlined item—the document file name—with the name of your document, probably My Tutorial 7. Then replace the second underlined item—the name to save the file under—to a name of your choice (perhaps My Automated Tutorial 7).

If you are using InDesign version 1.5, then you will also need to change the version number in the **tell application** statement from **2.0.1** to **1.5**.

Save the script after you have made your changes.

### Windows: Prepare the VBScript

Next, we will prepare the script which will control the automated importing process. (Macintosh users should skip the next subsection.)

- 14W** Open the Tutor\_7.vbs file in the Tutorial folder by right clicking on it and selecting **Edit** or by opening it from within any text editing application.
- 15W** Immediately select **File=>Save As...** from the menu and give the new script the name My VB Script.vbs. Be sure to save the file as **Text** format.

The existing script will require only minor changes for use with your document:

```
...  
' Open our demonstration document  
myInDesign.UserInteractionLevel = idDontDisplayAlerts  
myInDesign.Open (dataFolder & "Tutor 7.indt")  
  
...  
myInDesign.ActiveDocument.SaveAs (dataFolder & "Automated Tutorial 7.indd")  
myInDesign.ActiveDocument.Close
```

In these excerpts, we specify the InDesign document's name, and then later save the output file to a different name.

The underlined parts of the script will need to be modified for use with your document.

- 16W** Replace the first underlined item—the document file name—with the name of your document, probably My Tutorial 7.indd. Then replace the second underlined item—the name to save the file under—to a name of your choice (perhaps My Automated Tutorial 7).

Save the VBScript after you have made your changes.

### **Perform the Automated Import Operations (Windows and Macintosh)**

You are now ready to perform the automated import.

- 17** Run your completed script, in the same manner as before. When it has completed, open the new document in InDesign to verify that everything worked properly. If there are any problems, compare your file to the file Automated Tutorial 7 in the Tutorial folder.

This script imports the same data file into the document twice, probably not something you would want to do in real life. However, it is very common to want to import a large data set via a number of smaller batches. This topic will be discussed in more detail in Chapter 12.

## ***Congratulations!***

You have completed the final InData tutorial. The next several chapters of this manual cover data importing with InData, from the very basics in Chapter 4 to advanced prototype capabilities in Chapters 6 through 10 and more about automated document creation in Chapter 12.





# 4

## Basic InData Operations

This chapter covers InData's basic features, expanding on the concepts presented in the tutorials. The first section contains a brief overview of the product to enable experienced users of InDesign to get started using InData right away. If you completed the InData tutorials in Chapter 3, you may decide to skip the first section of this chapter.

### A Quick Overview of InData

InData is used to import and format data exported as text from database and spreadsheet applications into InDesign documents. (Note that InData *cannot* read the original database or spreadsheet application files.) Data for importing may also be created using a word processing program, or even created and published entirely within InDesign.

There are three steps required to format such data using InData:

- 1 Create a document in InDesign, including an InData **prototype**, which specifies exactly how each incoming data record is to be placed and formatted as it is imported.
- 2 Designate the prototype.
- 3 Choose an import option from the InDesign **InData** menu, set any necessary import parameters, and tell InData to begin importing.

The remainder of this section will look at each of these steps individually.

### Prepare the InDesign Document

Setting up the InDesign document into which the data will be imported is the first step in the process. You will need to create a new document, and then plan and set up its layout. This generally includes these important components:

- ◆ The number, size, placement and other attributes of text frames on each document page. This is accomplished by setting up one or more master pages for the document.

- ◆ The font, size and paragraph formats for document text. This is often accomplished by setting up style sheets specifying the desired character and paragraph attributes.

In general, the text frame into which you plan to import data records should be part of the automatic text thread in the document. That way, when imported records fill one page, a new page will automatically be created by InDesign (thanks to InFlow).

You can verify that a text frame on a document page is part of the automatic text thread by placing the text cursor within it, and then clicking on the **Object** menu. If the final item in the menu, **Default Auto-Flow Thread**, is checked, then the frame is part of it. Note that the item will be disabled unless you are at a master page.

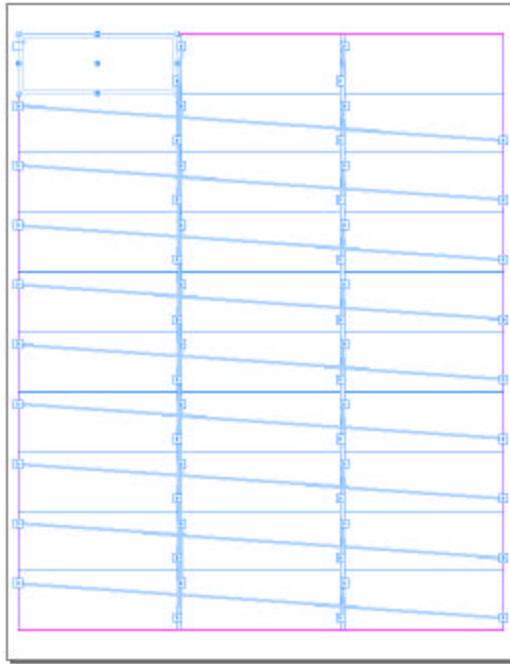
The main text frame on the page is automatically part of the automatic text thread when you create a new document with **Master Text Frame** checked. This alone is almost always sufficient for documents not requiring multiple text frames to achieve its desired page layout.

In other cases, you will need to create additional text frames by hand on the master page and then thread them into the automatic text thread manually. For example, to create a document for address labels, open a new document and leave **Master Text Frame** unchecked. Next, create text frames on the master page of the same size as an individual label, and place them in the same positions as the labels themselves are arranged on a sheet. Then, place the insertion point inside the first box, and select **Object=>Default Auto-Flow Thread**. Finally, thread the output port of that frame to the input port of the next frame, and continue the process until all of the text frames are connected.

The following illustration shows how the complete master page might look. (You can view text threads via **View=>Show Text Threads**).



You can examine the first of these sample documents yourself. It is located within the InData Samples folder: Avery 5162 in the Avery Labels subfolder.



## Create an InData Prototype

Once the document's structure has been set up, we next create the InData ***prototype***. An InData prototype is a normal text story that defines how each field in each record from the data file is to be positioned and formatted as it is imported into the InDesign document.

The prototype is typically placed in a text box located on the pasteboard beside page one of the document (although there are other options). Do *not* place the prototype on the document's master page.

The first line of a prototype is typically a **fields** statement, which is used to assign symbolic names to each field of the data records. The statement consists of the **fields** keyword, followed by a comma-separated list of field names, all enclosed in chevron marks—« and »—which must surround all InData prototype statements and expressions.

Here is a simple **fields** statement:

```
«fields fname, lname, title»
```

The data file this **fields** statement describes has three fields per record, which are assigned the names **fname**, **lname**, and **title** for the purposes of the prototype. These names need not be the same as the actual field names in the originating appli-

cation. Rather, they are completely arbitrary, and may be chosen as desired by the user.

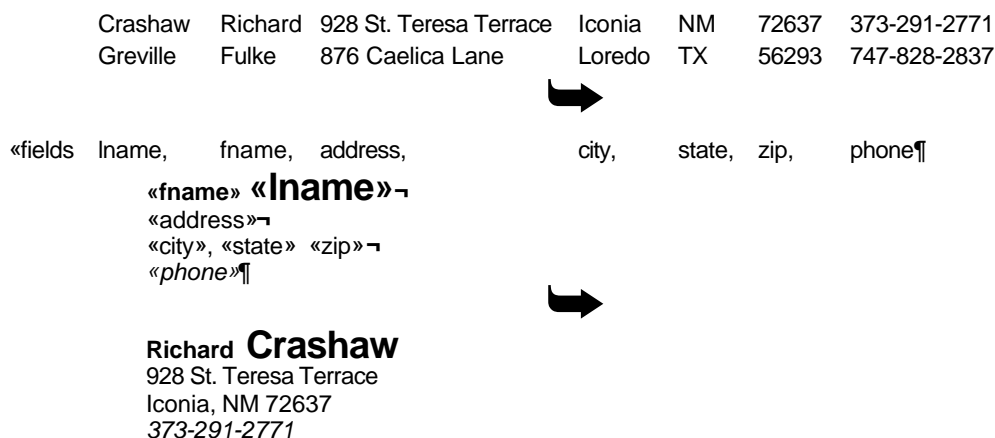
In fact, a **fields** statement is not technically required in a prototype. The automatic names **a** through **z** are always assigned to the first twenty-six fields of the data, regardless of whether there is a **fields** statement in the prototype or not. Thus, the automatic field name **b** is equivalent to **lname** within the previous prototype.

## Specifying the Formatting of Incoming Data

The remainder of the prototype consists of a sequence of literal text and **field placeholders** and other InData statements. Field placeholders consist of the field name—as defined in the **fields** or the letter corresponding to the field—enclosed in chevron marks: «*fieldname*».

These field placeholders specify the placement and formatting for the actual data records of the data file. Incoming data fields are laid out in the order given by the prototype, and each field's contents is automatically formatted in the same way as its corresponding field placeholder. All character formatting—including any character styles—of the field placeholders is carried over to the formatted data records. Paragraph settings—including the paragraph style—are also carried over from the prototype to the corresponding paragraphs of the formatted data. InData thus not only processes the data records and arranges them as you like, but also formats them as you instruct it to. Rather than having to format dozens or thousands of records by hand, you format the prototype *once*, and InData does the rest. And, if you use styles to specify character and paragraph formatting, you can experiment with, and change your mind about, the way records are formatted even after the fact, again in one simple step.

This process is illustrated in the following diagrams; note how corresponding fields in each successive record are constructed and formatted identically in the InDesign document.



**Fulke Greville**

876 Caelica Lane  
Loredo, TX 56293  
747-828-2837

We've expanded the spacing within the **fields** statement to make matching up the field names and the corresponding data fields easier.

InData requires that the character formatting be consistent within each individual field placeholder, including the enclosing chevron marks (or chevron mark and paragraph mark). This makes sense since you would be requesting ambiguous character formatting for a field if you changed the character attributes in the middle of a placeholder.

**Designating the Prototype**

The second step in the import process is designating the prototype: informing InData as to its location. When the prototype is placed in a text box on the pasteboard, this is accomplished by clicking in that text box with the text tool and then selecting **InData=>Use Story as Prototype** from the InDesign menu. A checkmark will appear next to this item whenever the prototype's text story is current.

We'll look at other ways to specify the prototype's location later in this chapter.

**Importing Data**

Prior to importing data, you must place the text cursor into the (presumably empty) text frame into which you want the formatted records to be placed. If your target location is a non-empty text frame, then the imported records will replace any text that is selected within the target text frame.

The data import process itself begins by choosing one of the **Import...** selections from the **InData** menu. The simplest case is to select **Import from File...** When you do so, InData will present a dialog asking for the data file to import. Once you select the data file, the InData control panel will appear on the screen (discussed and illustrated later in this chapter).

From the InData control panel you may specify the data file format (via the **Data...** button), the desired starting and ending records (via the **Range...** button), and the screen update frequency (via the **View...** button), in addition to actually starting the import process by pressing the **Start** button.

These basic steps are elaborated in more detail in the InData tutorials in the previous chapter.

**Reimporting Over Existing Data**

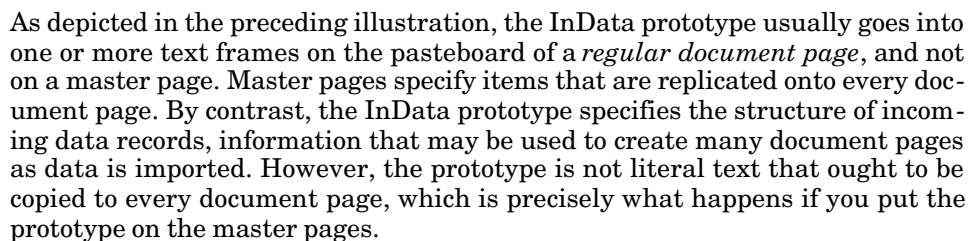
In general, InData will replace all selected text in the target text thread with the imported and formatted records. This has the following consequences in practice:

- # InData Prototype Fundamentals

## Where to Put the Prototype

document window

*text frame on pasteboard*



Placing the prototype into a pasteboard text frame on the first document page has several advantages over locating it within the text frame containing the main text thread:

- ◆ It makes it easy to re-import your data: once you've imported some data, you can just edit the prototype on the pasteboard, select the entire target story, and import again over the previously-imported data, all without copying and pasting the prototype.
- ◆ It lets you prepare the prototype in a “congenial” environment. Complex prototypes can be quite lengthy, much longer than any individual formatted record. Using this method, you can create the prototype in larger text frames that accommodate the extra length without spilling over onto multiple pages. Using a larger text frame for the prototype than for the imported data has no effect on the final results.
- ◆ In a related vein, you'll often use “new frame” characters in your prototype to force page or column breaks. Creating the prototype in several linked frames on the pasteboard allows you to keep the entire prototype in one place, rather than having it spread out over several pages.

You may select a text frame and designate it as the prototype with the **Use Story as Prototype** option as many times as you like. The most recently designated text frame is the one that will be used when an import option is chosen. This means you could create several alternate prototypes, choosing the appropriate one just before any given import operation.

Anytime you select the text frame currently designated as the prototype, the **Use Story as Prototype** menu item will be checked when you pull down the **InData** menu.

### Alternate Prototype Placement

Prototypes may also be placed directly within the text frame that is the target for the imported records (this was the method we used in the tutorials in Chapter 3). In this case, designating the prototype is done slightly differently. If the prototype is the only text within the target text frame, then you may simply place the cursor within that story, without selecting any text, and the entire story will be replaced by the import operation. If, on the other hand, there is any additional text within the target text frame that is not part of the prototype (a title or table heading, for example), then you must select the prototype portion of the story, including any trailing carriage returns or paragraph marks, prior to selecting an import menu option. Then, only the selected text will be replaced when data is imported.

The following example will illustrate not only the highlighting method of selecting the prototype, but also the fact that InData need not be used only for the highly regular, repeating documents, containing large numbers of records that we have considered so far. It may be used anytime you wish to format data in a com-

plex way. For example, InData may be used to create tables from data stored in spreadsheets (or in text data files, or even just in a text frame on the InDesign pasteboard) which take full advantage of InDesign's sophisticated typographic features. Consider the following table:

Romar V Sentence Development Level (0-14)				
Sector 42,A9X				
Common Name	Species	First Survey (c. 2200 C.E.)	Second Survey (c. 2500 C.E.)	Projected Level at 5000 C.E.
man	<i>homo sapiens</i>	20	400	??
dog	<i>canis domesticus</i>	15	50	60
wolf	<i>canis lupus</i>	10	20	5
cat	<i>felis domesticus</i>	5	40	40
bear	<i>ursa major</i>	10	20	3
bald eagle	<i>a. symbolotus</i>	10	40	.5
condor	<i>condoria euro</i>	2	1	0
beaver	<i>bustus mamalia</i>		30	20
tiger	<i>felis tigens</i>	45	23	2
cockroach	<i>pestus mostest</i>	300	600	900
elephant	<i>pachypus primo</i>	45	23	5
blue whale	<i>ostacus maximus</i>	5	2	.01

The reversed type effects are beyond the capabilities of most spreadsheet programs, and InDesign paragraph styles cannot specify multiple character formats within a single paragraph; the boldface and italic styles would need to be applied by hand to each line of the table. However, InData can handle all of these styles automatically.

Here is the prototype structure (minus the header lines):

```
«fields name, species, s1, s2, pj»
«name» «species» «s1» «s2» «pj»
```

Note that the **fields** statement follows the header lines on the page. To import data into this prototype, it is necessary to use the selection method; the whole prototype must be selected since the header lines should appear only once. If the cursor were simply placed in this text frame before importing, every imported record would be preceded by the header lines.

Romar V Sentence Development Level (0-14)				
Sector 42,A9X				
Common Name	Species	First Survey (c. 2200 C.E.)	Second Survey (c. 2500 C.E.)	Projected Level at 5000 C.E.
«fields name, species, s1, s2, pj»				
«name»	«species»	«s1»	«s2»	«pj»



Note that the final paragraph mark is included in the selection, since it is part of the prototype. It ensures that each record of data becomes a separate line in the table. Even though there are only a relatively small number of records of data in this example, InData is still of great help in ensuring consistent, accurate formatting.

## Choosing Field Names

As we've noted, field names are completely arbitrary, and need not have any relationship to the field contents or the actual field names in the original application database (or spreadsheet), although it's generally a good idea to make them similar. Field names may be up to 1,024 characters in length and must consist entirely of letters (defined to include the underbar—aka underscore—character) and numbers, and must begin with a letter.

The table below lists examples of legal and illegal InData field names:

LEGAL	ILLEGAL
lastname	last name
verylongname	very-long-name
_very_long_name	
Fahrenheit451	39steps
MiddleInitial	-balance
city_state	city,state

Field names are not case sensitive; any combination of upper- and lower-case letters may be used to refer to the same field. Thus, a field named **last** may also be referred to by the forms **Last**, **lAst**, **LAST**, **lasT**, and so on. (It's wise, however, to choose one particular and meaningful capitalization and stick with it.)

Although most of the prototypes in this manual will contain one, the **fields** statement is completely optional for data files containing less than 27 fields. The first 26 fields in any data file are always automatically assigned the names **a** through **z**. However, if the data file contains more than 26 fields, and the prototype does not contain a **fields** statement, then there is no way to access the 27th and later fields in each record (which in many cases poses no problem).

Thus, all three of the following prototypes are exactly equivalent and will produce the same results when used to import a data file:

```
«fields last, first, addr, city, sta, zip»
«first» «last»
«addr»
«city», «sta» «zip»

«b» «a»
«c»
«d», «e» «f»
```

```
«fields first, last, addr,ci,sta¶
«last» «first»¬
«addr»¬
«ci», «sta» «f»¶
```

*Looks tricky, but it's still the same 2 fields...*

The third example illustrates two important points about prototypes. First, it underscores the fact that the defined field names are arbitrary. It names the first field in each record—holding the person's last name—**first**, and names the second field **last**. However, since it still places their respective field placeholders in the same order on the first line of each record; the second field in each record comes first, followed by the first field. Thus, the final output will be the same as for the first prototype, which uses more intuitive (and sensible) field names.

Second, it illustrates the fact that you can mix automatic and user-defined field names within the same prototype, in this case by using a **fields** statement that specifies only the first five fields in each record. The prototype must use the automatic name **f** to refer to the sixth field, since it didn't name any field past the fifth.

## Skiping One or More Fields in the Data File

**Fields** statements need not contain a field name for every field in the data file. For example, the following **fields** statement doesn't name the fourth field in each record:

```
«fields last, first, initial, , room, phone»
```

All fields are always imported, however, so it is still possible to refer to the fourth field of the data file in the rest of the prototype by using its automatic name, in this case **d**:

```
«last», «first» «initial»¬
«phone»¬
«d» «room»¶
```

Data fields at the end of a record may be ignored without any special handling; thus, the previous **fields** statement will assign names to the first, second, third, fifth, and sixth fields in each record regardless of the actual number of fields in the records in the data file.

## Inserting Fields More Than Once

Imported fields may be used as many times as desired within the prototype. There is no restriction that each be used only once. Similarly, fields need not be used at all even though they are defined in the **fields** statement.

For example, if you wanted to create two address labels for each record in a data file, you could use a prototype like this one:

```
«fields last, first, initial, addr, city, state, zip»  
«first» «initial». «last»¬  
«addr»¬  
«city», «state» «zip»⌵  
  
«first» «initial». «last»¬  
«addr»¬  
«city», «state» «zip»⌵
```

The character at the end of the first label is a next frame character (Shift-Enter), which forces the second label into the next text frame. Similarly, the new frame character at the end of the prototype will force the subsequent label created by the next data record into the next text frame.

A similar technique could be used to create two or more slightly different labels for each incoming data record.

## Field Placement Flexibility

Fields also need not be placed into the document in the order in which they are defined in the **fields** statement (and that they appear in the data file). They may be placed in any order, regardless of their actual position in original data records.

In general, the **fields** statement merely defines names for the corresponding data file fields. It does not prescribe or control how those fields are imported or placed.

## Missing and Empty Fields

If some records in the data file do not contain information in every field, then the corresponding position for that record in the final output will be empty (blank). The rest of the prototype will be completed normally. For example, the following formatted record has no data in its **city** field:

```
Smith, John  
124 Wayside Heights Rd.  
, TX 67344  
111-222-3344
```

Methods of handling empty fields in the imported data are covered in Chapters 3 and 6.

## When to Omit the Right Chevron Mark

You may have noticed that **fields** statements in example prototypes often omit the final » mark. The reason for this is illustrated clearly by the following example. The illustration shows the different document forms resulting from two prototypes which differ only by the presence or absence of the closing chevron mark on the **fields** statement.

PROTOTYPES:

«fields last,first,ext¶		«fields last,first,ext»¶	
«last», «first»»	«ext»¶	«last», «first»»	«ext»¶

SAMPLE OUTPUT:

<b>Crashaw, Richard</b>	<b>2213</b>	¶	
<b>Grenville, Fulke</b>	<b>2543</b>	<b>Crashaw, Rlchard</b>	<b>2213</b>
<b>Howard, Henry</b>	<b>8732</b>	¶	
<b>Jonson, Ben</b>	<b>2311</b>	<b>Grenville, Fulke</b>	<b>2543</b>

... and so on ...

In the right prototype, the **fields** statement is terminated by the » mark. The paragraph mark—which we’ve underlined—is a literal character in the prototype, exactly like the comma after the last name. Thus, it is reproduced as the first character in every imported record, producing the extra spacing between records in the final document.

By contrast, the **fields** statement in the first prototype is terminated by the paragraph mark itself; here, the paragraph mark serves the same function as a » mark. Thus, it is part of the **fields** statement and not a literal character in the prototype, and accordingly, it does not appear in the formatted records in the resulting document.

In general, any closing chevron mark may be replaced by a paragraph mark (created by entering a carriage return). However, if a closing chevron mark is used, and a new paragraph is not desired, then the next portion of the prototype should be placed on the same line with no intervening text. For example, if a **fields** statement is closed with the » mark, then the initial line of the formatting specifications should be placed in the *same* paragraph with it, as in this example:

«fields first, last, title, room, ext»«last», «first»»	«title»¬
Ext. «ext» (Room «room»)¶	

Replacing the closing » mark with a paragraph mark may often be helpful in making prototypes clearer and easier to understand. Further examples of these considerations will be given later in this manual.

## More Examples of Formatting Prototypes

InData’s advantage over the built-in report generation capabilities of database and spreadsheet packages is its ability to bring the publishing power of InDesign to your data. InData may be used to format the same data in many different ways, limited only by your imagination and available time. The prototypes to follow illustrate some alternate treatments of some employee name and address data, having the following fields:

- ◆ last name
- ◆ first name

- ◆ address
- ◆ city
- ◆ state
- ◆ zip code
- ◆ title
- ◆ department



We've already seen that address labels are easy to format using InData. The basic procedure when creating a label document is to create text frames on the master page for each label on the sheet and then to enter the prototype into the first frame on page one of the document (don't make the mistake of placing the prototype on the master page!). Here is a sample prototype:

```
«fields last,first,addr,city,state,zip,title,dept»
«title» «first» «last»¶
«addr»¶
«city», «state» «zip» ¶
```

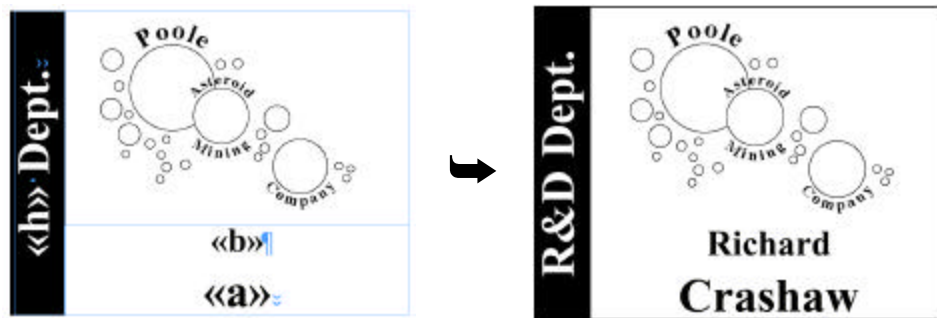
Notice the InDesign “new frame” character at the end of the prototype (created by typing Shift-Enter). Placing it at the end of the prototype ensures that each record will be in a separate text frame, corresponding to its own label. Set text frame vertical alignment to **Center** (by selecting the frame and then using the **Text Frame Options** command from the **Object** menu) to center the text vertically on the label.

Form letters are also easily produced with InData. For example, the prototype below creates an invitation for each person in a data file:

<p><i>The Board of Directors¶ of the Poole Asteroid Mining Company¶</i></p> <p><i>cordially invite¶</i></p> <p><i>«g» «b» «a»¶</i></p> <p><i>to our 50th Anniversary Celebration¶</i></p> <p><i>Saturday, June 12, 2077¶ 2:00 to 6:00 p.m.¶</i></p> <p><i>The Beachfront Hotel ¶</i></p>		<p><i>The Board of Directors of the Poole Asteroid Mining Company</i></p> <p><i>cordially invite</i></p> <p><i>Mr. Richard Crasshaw</i></p> <p><i>to our 50th Anniversary Celebration</i></p> <p><i>Saturday, June 12, 2077 2:00 to 6:00 p.m.</i></p> <p><i>The Beachfront Hotel</i></p>
--	--	--

The majority of this prototype is literal text, with the last name and first name fields from the data file inserted in the fourth line (the second and first field per record, respectively). Like the previous example, the prototype concludes with a next frame mark to ensure that each invitation is placed properly on the page.

The same data file can also be used to create signs for office doors. Signs could be printed on heavy card stock and then cut to fit existing holders. Here is one possible prototype, along with one of the signs it produces (both greatly reduced in size):



As you can see from this example, InData prototypes may be used with nearly any set of InDesign features, like the shaded and rotated text frames illustrated.

### Forcing Text to the Next Column, Frame, Page and Even/Odd Page

We've seen several different ways to force text to start in the next column or next text frame in this and the previous chapter:

- ◆ Follow it by a new column character (Enter). This will force text to the top of the next column in multi-column text frames and into the next text frame in single column text frames.
- ◆ Follow it with a new frame character (Shift-Enter), which always begins subsequent text in the next text frame.
- ◆ If you want a break before a given paragraph, you can make the paragraph's **Space Before** setting a value larger than the depth of the text frame. For example, to force each record on to its own address label without using a final new frame character within the prototype, set the **Space Before** value for the paragraph containing the first line of each label to some value that is longer than an individual label (6" for example).

And there are several other break-control methods you can use with InDesign:

- ◆ If you want a break after a given paragraph, you can make the paragraph's **Start Paragraph** setting (in the **Keep Options**) one of **In Next Column**, **In Next Frame**, **On Next Page**, **On Next Odd Page**, **On Next Even Page**.

- ◆ InDesign also has special break characters for next column, frame, page, even page, and odd page, and you can use the appropriate break character for your situation in an InData prototype.

## The InData Menu

We will now turn to a brief consideration of each of the items on the **InData** menu.

**Import from File...**

Import data records from an external file.

**Import from Clipboard...**

Import data records that are currently residing on the system clipboard.

**Import from Pasteboard...**

Import data records residing in a unique text story on the pasteboard.

**Make Header/Footer...**

Create a mark reference for use in generating automatic headers and footers.

**Update Headers/Footers...**

Update any relevant headers and footers on the current story to match the current state of edited, imported text.

**Use Story as Prototype**

Designate the currently selected text story as the prototype.

**Name Story...**

Assign a name to a text story within the document for later AppleScripting.

**Name Substory...**

Assign a name to a subset of a named story for later use with AppleScripting.

**Find Story/Substory...**

Locate a specified story and possibly substory within the current InDesign document.

**Preferences**

Edit InData preferences. Contains a submenu consisting of **Data...**, **View...**, **Range...**, and **General...**, which are used to set specific types of preferences.

**About...**

Display an informational window describing this copy and version of the InData plug-in.

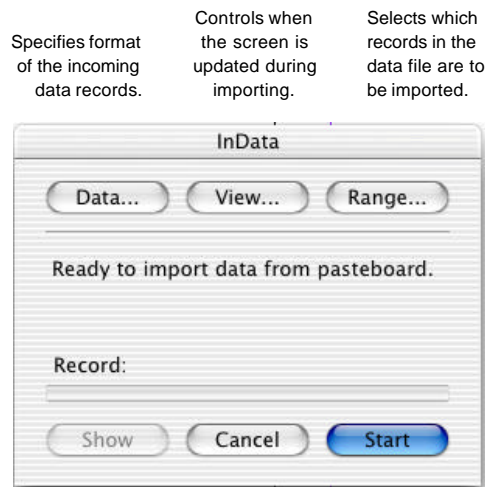
The various InData menu items will be discussed frequently throughout this manual.

## Data Import Options

Normally, InData imports records from an external data file, but there are other potential source locations for the raw data records. For example, data may come from the system clipboard. You import clipboard records by simply selecting **Import from Clipboard...** rather than **Import from File...** from the **InData** menu. Any formatting information in the data—italics for example—is ignored and will *not* be carried over into the formatted records. Rather, as always, each field's format will be the same as that of its field placeholder. If there is not currently any text on the system clipboard, then this menu item will be dimmed.

Data records may also reside in a text frame on the pasteboard. If you choose the **Import from Pasteboard...** option from the **InData** menu, then the data records will be assumed to be in a text frame on the pasteboard of the current spread. This text frame must reside entirely on the pasteboard and be the only text frame placed there (except for a text frame which has already been designated as holding the prototype). As for text imported from the clipboard, all formatting information in the data is ignored.

Once an import option is selected and a data file is specified (if applicable), the InData control panel appears on the screen:



Specifies format of the incoming data records.

Controls when the screen is updated during importing.

Selects which records in the data file are to be imported.

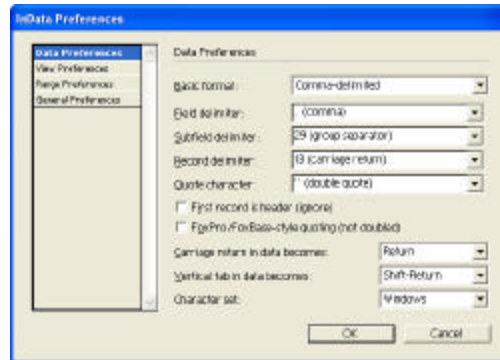
Status messages appear in this area.

Import progress is indicated by the bar and record number.



We'll look at each of the related dialogs individually in the sections that follow.

## Specifying the Data Format



The **Basic format** field specifies the format of the data. Selecting a basic format automatically fills in the other fields in this dialog.

Check this box to skip the first data record.

The **Data...** button is used to bring up the **Data Preferences** dialog. The most important settings in this dialog are the **Basic format** field and **First record is header** setting.

The **Basic** format field is a pop-up menu listing the most common exported data file types exported by database and spreadsheet programs. This menu includes the following selections:

### Tab-delimited

Fields are separated by tabs within a record, and records are separated by carriage returns. Fields containing tabs or returns are entirely enclosed in double quotation marks, and all double quotation marks are doubled.

### Comma-delimited

Fields within a record are separated by commas, and records are separated by carriage returns. If a field contains a comma or a carriage return, it is entirely enclosed in double quotation marks, and all double quotation marks are doubled.

### MS Word™ merge comma-delimited

The comma-delimited format exported by Microsoft Word (regular comma-delimited with a header record that is automatically ignored).

### MS Works™ tab-delimited

The tab-delimited format exported by Microsoft Works' database/spreadsheet application (regular tab-delimited with a header record that is automatically ignored).

**Custom**

User-defined data format: you must fill in any special values for the other fields in the dialog. See Chapter 13 for details.

Select the format from the pop-up menu corresponding to the one you used when exporting your data from its original application program. We recommend importing files in the tab-delimited format, so select this format when exporting your data if your application offers it as one of its choices.

The **First record is header** check box should be checked if your application program outputs a header record (usually containing the names of the fields that were exported) in exported data files. Choosing this option instructs InData to skip this first record in the data file. (Note that InData cannot process this record to assign field names.)

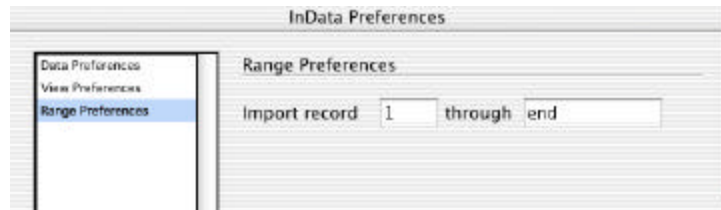
Select the **FoxBase/FoxPro-style quoting (not doubled)** check box in the data preferences dialog when using FoxBase or FoxPro data snapshot output. This option prevents double quotes (") within fields from being interpreted as a single quote (').

For the vast majority of import operations, choosing one of the standard formats is the correct thing to do. If you have exported data from your original application program using a built-in export option, then it is very unlikely that you will need to change any of the settings.

You may override any of the standard settings for any format by entering specific values into the series of fields on this dialog. Values are entered either by choosing them from the pop-up menus (which attempt to list every nonprinting value a setting might take on), or by directly typing them into the spaces provided. Numeric values of two or more digits are assumed to be ASCII codes for the desired character.

Changing a field's contents to a blank (or beginning with a blank) disables that field, and informs InData that all corresponding characters should be treated as literal data. For example, entering a blank into the **Quote character** field disables the double quote character as a quote character, and InData will treat all double quotes found as simply part of a data field.

## Importing a Range of Records



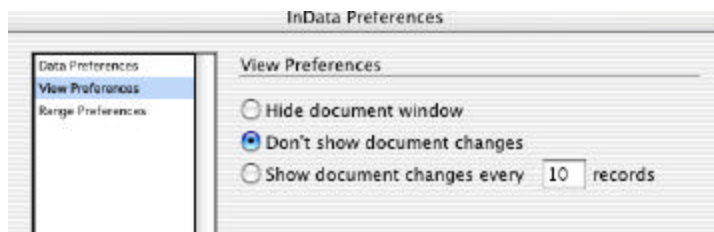
The **Range...** button on the InData control panel brings up the **Range Preferences** dialog. It may be used to specify the range of records to import from the data file (or other data source). Normally, all records are imported. You may specify a range of records to import by entering the beginning and ending record numbers into the frames in the second line. For example, to import only records 14 through 45, inclusively, enter **14** into the frame following the word **record** and enter **45** into the frame following the word **through**.

Record numbering begins at 1. The keyword **end** may be used in place of the ending record number to specify that all records from the beginning record are to be imported. **Last**, **all**, and **final** may also be used as synonyms for **end**.

If the number entered as the final record is greater than the number of records in the data file, no error occurs. InData will stop processing once the last record in the data file has been imported.

If the **First record is header** check box in the **Data Preferences** dialog is checked, the records specified here are what is to be imported after skipping the first record. If you only need to skip the first header record in your data file, then use the **First record is header** check box in the **Data Preferences** dialog, rather than specifying an explicit record range of **2** through **end**.

## Document View Options



The **View...** button on the InData control panel brings up the **View Preferences** dialog. It may be used to control the rate at which the screen is updated. Normally, the screen is not updated until all data records have been imported and processed (unless you're importing pictures, in which case InDesign will spon-

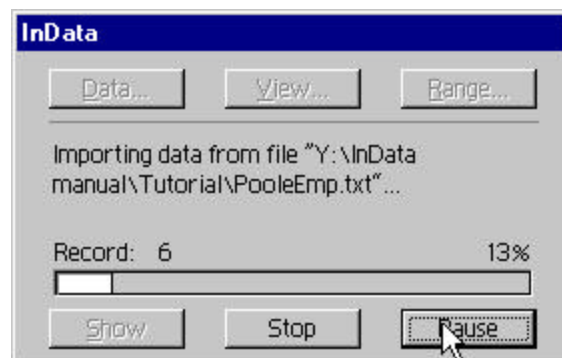
taneously update the screen after every picture frame insertion); this behavior is equivalent to the second radio button in this dialog, labeled **Don't show document changes**.

You may change this default screen update behavior if you like by selecting either of the other buttons in this dialog. The third button, **Show document changes every \_\_ records**, determines how often the screen view is updated during data importing. For example, a setting of **20** means that the screen will be updated each time 20 records are imported. Note that frequent screen updating will slow down the import process.

The top button, **Hide document window**, will cause InDesign to show absolutely no window updates during the entire import operation. It is the fastest import mode, particularly when importing pictures along with text.

The InData control panel's **Show** button may be used to request an immediate screen update at any point in the import process.

## Controlling Data Importing



You begin data importing by pressing the **Start** button on the InData control panel. Once data importing has begun, the **Start** button changes to **Pause**, and the **Cancel** button changes to **Stop**. Clicking on the **Pause** button will temporarily pause data importing and formatting, and give you an updated view of the last record imported. (There may be a slight or lengthy pause while InDesign computes the updated screen image.)

When InData is paused, the button's name changes again to **Continue**; clicking on the **Continue** button will resume processing.

The **Stop** button at the bottom of the control panel immediately aborts InData's operation at whatever record is being imported when you press it.

The **Show** button is used to request an immediate screen update. When you press it, InData will update the screen image regardless of the settings in the **View Preferences** dialog.

## Prototype Errors

When you select an import option from the InData menu, the first thing InData does is to scan the prototype for errors. If there is an error in the structure of your prototype, InData will inform you of this and abort the import operation. No data records will be imported. Instead, an error message will appear in an alert dialog, and the incorrect part of the prototype will be highlighted in your document when you dismiss the alert.

If the structure of your prototype is correct, but an error occurs while processing your data, importing will cease at that point. If you placed the prototype in a separate text frame on the pasteboard, then you may simply go there and edit it. Once you've corrected the problem, delete the imported records, and then try the import operation again.

On the other hand, if the prototype was in the target text frame, you may end up with only part of your data (or none at all) in the final document and your prototype gone. Unless you are importing the data from clipboard, this is still no cause for alarm, however. When it can, InData always copies the prototype to the clipboard before importing any data. Thus, you can use the following steps to recover from these kinds of errors:

- 1 Place the cursor in the position where you want the prototype to go (usually at the beginning of the document) and then select **Paste** from the **Edit** menu. The prototype will be pasted at the insertion point.
- 2 If you do not want to save the partially formatted data, then select all of it and delete it from the document.

Alternatively, you may wish to create a new document and paste the prototype into it in order to save both the prototype and the partially completed document.

*Warning:* If you are going to import data from the clipboard, then we recommend that you always place the prototype in a different text frame from the target text frame which will receive the data. Otherwise, if you failed to save the file before importing data, there is no way to recover the prototype once data has been imported.

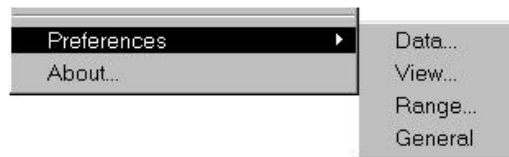
For specific solutions to common importing problems, see Chapter 14.

## Setting InData Preferences

All of the settings we've seen in the previous section can be set on a permanent basis, either for a single document or globally for all InData operations. The InData menu's **Preferences...** option allows you to set defaults for InData.

InData preferences work the same way as InDesign's preferences. Whenever preferences are set while a document is open, the settings apply to that document alone. However, if they are set without any open document, they apply to every document that does not have its own specific preferences. Of course, the settings specified as InData preferences may always be overridden for any import operation by explicitly changing the desired settings via the InData control panel.

Selecting **Preferences** on the **InData** menu produces a slide-off menu.



The **Data...**, **View...**, and **Range...** options bring up the same dialogs as the corresponding buttons on the InData control panel.

The **General...** option brings up the **General Preferences** dialog, with the following settings.

The **Default picture position** field indicates where and how imported pictures will be placed within their picture frames. It has the following options on its pop-up menu:

### Top Left

Place the picture's upper left corner in the upper left corner of its picture frame. (This is the default in InDesign; the same behavior results from the **Place...** option on the **File** menu.)

**Center** Center the picture in the picture frame.

### Center, size to fit

Size the picture to fit the picture frame exactly.

### Center, size to fit, w/o distortion

Size the picture to fit the picture frame, maintaining the picture's original proportions (aspect ratio), and then center it.

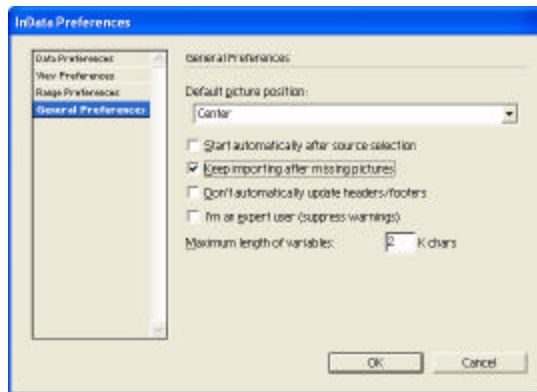
### Size Frame to Picture

Shrinks the anchored picture frame to fit its contents (once they are

imported), obeying any scaling and margin picture frame properties (see chapter 5), and ignoring any offset picture frame properties.

#### **Size to Fit Horizontally, then Size Frame Vertically to Picture**

Sizes the picture itself in its anchored picture frame to fit horizontally (i.e., to fill the frame in the “x” direction) once the picture is imported, sets the picture’s y scale to match its x scale (as determined above by making it fit horizontally), and then shrinks the anchored picture frame itself vertically to make it fit the contained picture.



The **Start automatically after source selection** button specifies whether the user needs to press the **Start** button on the control panel before data importing begins. If it is checked, then importing begins automatically as soon as InData knows where the data records are. By default, it is unchecked.

If the **Keep importing after missing pictures** box is checked, then InData will continue importing data records even if it cannot find one or more picture files and leave the corresponding picture frame(s) empty. By default, data importing ceases immediately when a picture file can't be located.

The **Don't automatically update headers/footers** check box controls whether headers and footers are updated automatically at the conclusion of an import operation. By default it is unchecked.

If the **I'm an expert user (suppress warnings)** box is checked, then the warning message that occurs when the imported data overflows its text thread and other warnings are not displayed. It is unchecked by default.

The **Maximum length of variables** field controls the maximum number of characters that you can place into a variable with InData's **put** prototype statement, in multiples of 1024 bytes (1K); by default, the length is 2K characters. InData variables are discussed in Chapter 10.





# 5

## Preparing Data for Importing

This chapter discusses methods of preparing data for use with InData. InData can import data exported from virtually any Macintosh or Windows spreadsheet or database application.

### Data File Formats

InData imports the following standard text file interchange formats:

***Comma-delimited fields:*** Field values are separated by commas within a record, and records are separated by ASCII carriage returns. Field values containing embedded commas are enclosed in double quotation marks. When exporting data to this format, some applications, such as FileMaker Pro, convert double quotation marks within field values to single quotation marks, but others double them properly.

This format is also called “comma format,” “comma text format,” “BASIC format,” and “CSV” (for comma-separated values).

***Tab-delimited fields:*** Field values are separated by ASCII tabs within a record, and records are separated by ASCII carriage returns. This format is also called “tab format,” “tabbed text format,” “text format with tabs,” and “TSV” (for tab-separated values). Double quotation marks are used to surround fields with embedded tabs and double quotation marks (which are doubled).

InData provides a built-in variation of the tab-delimited format, which automatically skips an initial header record, called ***Microsoft Works™ Tab-delimited*** format.

***Microsoft Word™ Merge File:*** This format is most associated with Microsoft Word, but it is also used by MacWrite II, for example. It is very similar to the comma-separated fields format, but includes an additional header record at the beginning of the data file containing the field names; this header record is ignored by InData and cannot be used to specify import field names.

It is also possible to define and use custom record formats. This topic is discussed in Chapter 13.

### Handling Carriage Returns within Fields

When exporting in some of these interchange formats, some application packages convert ASCII carriage returns within field values to ASCII vertical tab characters (e.g., FileMaker Pro does this in its merge file format, and Microsoft Word converts Shift-Returns to vertical tabs as well). By default, InData reconverts these vertical tab characters to hard carriage returns—the same character entered by pressing Shift-Return within InDesign—causing each field value to remain a single paragraph (or remain within a larger single paragraph also containing other text). You can override this behavior by specifying a different value in the **Data Preferences** dialog (see Chapter 4).

## General Exporting Procedures

The general procedure for preparing and exporting data files is outlined below.

- 1 With your original application program, open the file containing the data you want to import into InDesign.
- 2 If necessary, sort the data (or the subset of the data) you plan to export within the application program, using the appropriate key fields for the use you plan to make of it within InDesign.

For example, if you are preparing a phone directory in which you want all names to appear in alphabetical order, then sort the data by last name, then first name.

On the other hand, if your directory will be arranged by department, then sort the data with department as the primary sort field (and possibly last name and first name as secondary sort fields). Refer to your application program documentation for details on its sorting procedures.

- 3 Select the appropriate export option from the application's menus.

The name of the export option varies; some common names are **Export**, **Save As**, **Output To** and **Copy To**.

- 4 Specify what subset of the data you want to include.

In a database application, this means specifying which records and which fields are to be included. These two activities may be done in separate steps. For example, in FileMaker Pro, you specify which records to include before selecting the export command **Export...**, and you specify which fields to include in its **Specify field order for export** dialog. In a spreadsheet application, specifying the subset of the data means specifying what row and column ranges are to be included.

Again, depending on the application, this may need to be done before selecting the export command.

- 5 Select the appropriate data file format from those supported by your application.

“Tab-separated fields” is usually the best choice (note that it may appear under a slightly different name—check the list in the first section of this chapter).

- 6 Specify the name of the output file.

Be sure to choose a different name from your original application file. One convention is to give the data file the name of the original file with a suffix indicating the file’s format. For example, a tab-separated fields format data file exported from a database file named *Employees* might be called *Employees.tsv*. Another common practice is to give such files the extension *.TXT*.

- 7 Begin the export operation by clicking on the appropriate button in the application’s dialog (often **Save**, **OK** or **Output**).

Depending on your application, the steps above may need to be done in a different order. For example, specifying the name of the output file might precede specifying the file format, or specifying the subset of a spreadsheet to be included (perhaps by selecting the range you want) might need to be done before selecting the export command. Check the manual for your application’s specific details.

## Exporting Data from Database Applications: An Example from FileMaker Pro

FileMaker Pro is one of the most popular Macintosh database programs. This section goes through the export process for FileMaker Pro. The procedures for other database applications are similar, although the menu names vary and the dialogs look different.

- 1 Start FileMaker Pro and open the database file containing the data you want to publish with InData and InDesign.
- 2 If necessary, select and sort the subset of the database records you want to include.

This may be done either by using the **Find** command to select a subset of the database, or by using the **Omit** and **Omit Multiple...** commands to exclude certain records. Sort the records into the order in which you want them to appear in the InDesign document, using the **Sort...** command.

- 3 Select the **Export** command from the **File** menu.

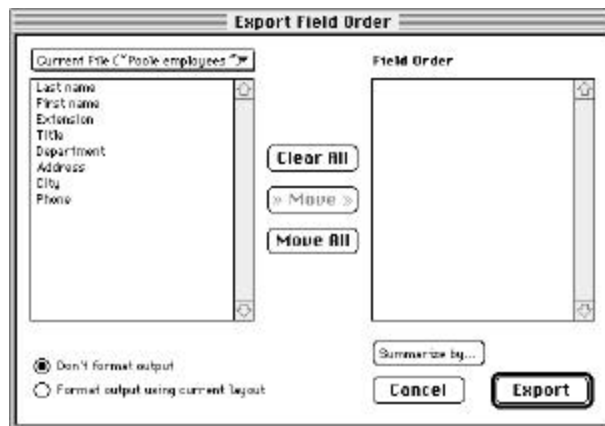
The **Export** dialog will appear. Use this dialog to specify the name and location of the export data file you are creating. You may choose any name you wish, but be sure to use a *different* name from that of your original database file.

- 4 Select **Tab-Separated Text** from the **File Type** pop-up menu.

We strongly recommend that you use this format when exporting from FileMaker Pro. If you use the **Comma-Separated Text** (comma-delimited) format instead, the following fairly severe restrictions will apply:

- ◆ You will lose the ability to access the subvalues in repeating fields.
- ◆ You will lose any embedded carriage returns in data fields.
- ◆ Field lengths will be truncated to 256 characters.

- 5 Click the **New** button. The **Export Field Order** dialog will appear:



You will use this dialog to specify the fields to include in the data file and their order within each record.

The left list frame initially lists all the fields in the database. When the dialog is exited, all of the fields listed in the right list frame will be exported, in the order in which they appear.

To designate a field for exporting, select it from the left list frame and then click the **Move** button. Select each field you want to export in turn, and continue this process until all of the fields you want are listed in the right list box.

- 6 We recommend that you select the **Don't format output** radio button below the field list box.
- 7 Click the **OK** button to close the dialog and begin data exporting. Exit from FileMaker Pro.

Your data file is now ready for importing into InDesign using InData.

### Handling FileMaker Pro Databases with Repeating Fields

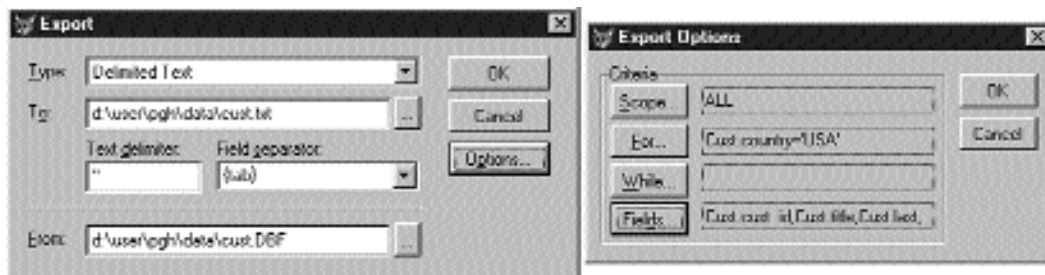
Multivalued fields—fields which can take on more than one value per database record—require no special handling when exporting them to InData and InDesign as long as the export file format is **Tab-Separated Text**. FileMaker Pro calls such fields *repeating fields*.

InData can access the separate values in repeating fields via its **subfield** operation (see Chapter 7).

## Exporting Data from Database Applications: An Example from Visual FoxPro

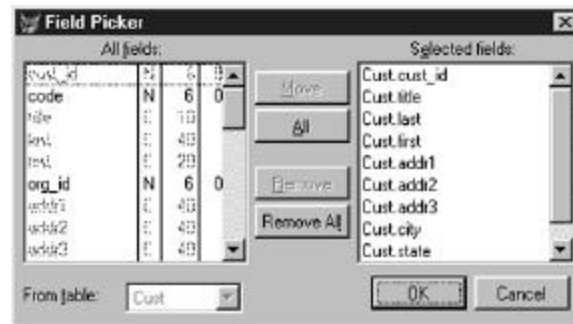
This section describes the export process for Visual FoxPro, using the Windows version as its example.

- 1 Start Visual FoxPro and then open the table from which you want to export data. One way to do so is to issue the **use** command in the command window.
- 2 Sort the data into the desired order. Often this can be accomplished by a **set order to** command referencing an existing index.
- 3 Select **Export** from the **File** menu. The dialog on the left will appear:



Use the **To** field to specify the output file's name and location. We recommend selecting the **Delimited Text** option from the **Type** popup menu and setting the **Field separator** to {tab} and the **Text delimiter** to a double quotation mark.

- 4 Click the **Options...** button to access the **Export Options** dialog (illustrated above on the right). Press the **Fields...** button in order to select the fields to be exported. The **Field Picker** dialog will appear:



Move each field that you want exported to the **Selected Fields** list in turn. Note that you can specify a different open database by selecting its name from the **From Table** list. Click **OK** when all of the fields you want are in the **Selected Fields** list.

- 5 Use the **Scope...**, **For...** and **While...** buttons to specify conditions which will select the subset of the data that you want to export. In our example, we have selected records from where the country is the USA.

Note that you will almost certainly need to include a join condition if you are exporting fields from more than one table.

- 6 Close the subordinate dialogs and click the **OK** button in the **Export** dialog when you are ready to export the records.

Your data file is now ready for importing into InDesign using InData. (Be sure to turn on the **FoxBase/FoxPro-style quoting (not doubled)** option in your InData data preferences.)

Note that the entire export operation could be performed from the command window with a **Copy To** command like this one (assuming the database is already open):

```
COPY TO Cust.Txt FIELDS cust_id, title, last, first, ... ;
FOR country="USA" TYPE DELIMITED WITH TAB ...
```

## Exporting Data from Spreadsheet Applications: An Example from Excel

This section goes through the export process for Excel. The procedures for other spreadsheet applications are similar, although the menu names vary and the dialogs look different.

- 1 Start Excel and then open the spreadsheet containing the data you want to publish using InData and InDesign.
- 2 If you want to export only a portion of the spreadsheet, then you have two options:
  - ◆ You may remove all of the unwanted cells from the spreadsheet—or a copy of it—before exporting.
  - ◆ Or, you may select only those rows and/or columns you want from the InData control panel, or from the prototype itself.

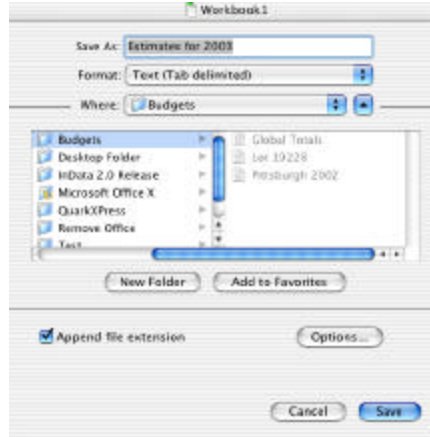
Extra columns are ignored by leaving those fieldnames null in the InData **fields** statement (or by omitting them at the end, or by just not using them in your prototype). Extra rows are ignored by specifying starting and ending records via the InData control panel's **Range...** button. The latter operation is practical only if all the rows you want to keep are contiguous.

If you decide to remove extraneous data from the spreadsheet, do it at this point. Also do any sorting you need at this point.

- 3 Select **Save As...** from the **File** menu.
- 4 Select the export data file name and location in the **Save As** dialog.

You may choose any name you wish, but be sure to use a different name from that of your original spreadsheet file. We suggest naming the data file something like name.tsv where name is the name of the original spreadsheet.

- 5 Select **Text (Tab delimited)** from the **Save as Type** popup menu:



- 6 Click the **Save** button to close the dialog and begin data exporting. Exit from Excel.

Your data file is now ready for importing into InDesign using InData.

## Creating Data Files Manually

Data files for InData may also be created by hand with any word processing program or even within InDesign (in a text frame on the pasteboard). These data files may use either the tab-delimited or comma-delimited format type. When typing your data, separate fields by tabs, and enter a carriage return at the end of each record. Each record thus becomes one line (or paragraph, in a InDesign story). If a field value contains a tab or a comma, enclose the entire value in double quotation marks. If a field value contains a double quotation mark, then enclose the entire field in double quotation marks and double the double quotation mark. For example, typing "data with "quotes" embedded" produces the field value data with "quotes" embedded.

If you're using a word processing program, be sure to save the file in **Text Only** format; consult your word processing program's manual for details. If your word processing package has two kinds of **Text Only** formats (as does Microsoft Word), choose the text format *without* line breaks.

It is also possible to use InDesign to maintain a simple database. The data lives in a text story on the pasteboard, and it may be formatted with InData at any time. You may want to create a InDesign template consisting of the data and prototype. Then, any time you want to publish the data, you simply open the template, import the data from the pasteboard with InData, and produce a new InDesign document, leaving the original data and prototype unchanged for future use. When you want to update the raw data, open the template, edit the data, and then save the file as a template under its original name, replacing the old version.



# 6

## Conditional Data Importing

In this chapter, we'll look at InData's conditional importing facility. First, we'll consider some examples that commonly occur in prototypes, and then we'll turn to a discussion of conditional prototype statements in general.

Many times data files will contain records with some empty fields. For example, a database of names and addresses will often allow two fields for the address, but not all records will use both of them. When importing data like this, InData allows you to conditionally import a field, so that the field will only be included in the document if there is data in it.

The prototype below illustrates this feature:

```
«fields last, first, addr1, addr2, city, state, zip»  
«first» «last»  
«addr1»  
«—only import field addr2 if there is something in it»  
«if addr2 is not empty» «addr2»  
«endif» «city», «state» «zip»
```

The shaded part of the prototype tells InData to include the data in the field **addr2** and the following paragraph mark only if that field is not empty.

This example also illustrates the use of comments within InData prototypes. The fourth line of the prototype is a comment, which is ignored during the import process. When a *comment designator*, either a pair of hyphens or a single em-dash, appears within a prototype statement, everything after it is treated as a comment and ignored by InData until the close of the prototype statement (» mark, new paragraph, new box, and so on). For example, the following two prototype statements are treated identically by InData:

```
«if last is not empty»  
«if last is not empty — check for a valid, non-empty last name»
```

As illustrated, an em dash, “—”, may also be used as a comment introduction indicator.

Note that double hyphens and em dashes are *not* treated as comment indicators when they appear inside of quoted character strings within prototype statements, as in this example:

```
«if code <> "XJ7--12A"»
```

## if Statements

The basic form of an **if** statement is:

```
«if condition» statements «endif»
```

where *condition* is a test to be performed and *statements* are any prototype statements—often field placeholders and literal text—which are to be included for each data record *only if* the stated condition is met (true). In the first example in this chapter, the condition was **addr2 is not empty**, and the statement was **«addr2»¶**; the former checks whether the field **addr2** is empty or not, and the latter tells InData to include the contents of the field **addr2** and a paragraph mark (carriage return). This test occurs for each imported record, and the **addr2** field's contents and the paragraph mark are included *only* when that field is not empty.

It is important to note that the **endif** statement comes after the paragraph mark following the **addr2** placeholder; otherwise, two paragraph marks would result, regardless of the contents of field **addr2**. Thus, the paragraph mark is itself conditionally included in the imported text based upon the contents of the field **addr2**.

An **if** statement is not limited to selective inclusion of fields, however. For example, an **if** statement may also be used to specify alternate formatting of a field based on some condition. For example, the prototype below prints the **zip** field in 18 point type if the **state** field holds **CA** and in normal 10 point type otherwise:

```
«fields last, first, addr, addr2, city, state, zip»¶
«first» «last»¶
«addr»¶
«if addr2» «addr2»¶
«endif» «city», «state» «if state is "CA"» «zip» «else» «zip» «end if»¶
```

The **zip** field's contents will be inserted into each record in any case, but its formatting will depend on which particular field placeholder was selected, which in turn depends on the contents of the state field.

The preceding prototype also illustrates a couple of other points about **if** statements. First, as its fourth line illustrates, the **is not empty** keywords are optional in **if** statements testing only the presence of a single field: **«if addr2»** is equivalent to **«if addr2 is not empty»**.

Secondly, the preceding prototype also introduces the **else** statement, which precedes statements to be processed if the condition is not true. The general form of the **if** statement is thus:

«if condition» true-statements «else» false-statements «end if»

Both sets of statements may contain literal text, field placeholders, expressions involving literal text and field placeholders, and additional prototype statements (including nested **if** statements).

Finally, note that **endif** may also be written as two words: **end if** (as in the final line of the prototype). Both forms are equivalent. We will use the single word form throughout this manual.

## Constructing Conditions

As we've seen, one way to construct a condition in an **if** statement is to compare the contents of a field against some other value (including the empty string). The comparison possibilities include:

- ◆ a literal value;
- ◆ the contents of another field;
- ◆ the contents of the same field in the previous record;
- ◆ any expression (we'll discuss some examples of these later in this chapter).

Here are some examples of conditions:

<b>last = "Smith"</b>	<i>True if the <b>last</b> field is "Smith".</i>
<b>city &lt;&gt; old_city</b>	<i>True when the fields <b>city</b> and <b>old_city</b> contain different values.</i>
<b>dept &lt;&gt; prev dept</b>	<i>True when the value in the <b>dept</b> field in the current record differs from the one in the previous record.</i>

These examples introduce two new comparison operations using the symbols **=** and **<>**. The equals and not equals signs are examples of operators, which indicate how the two items in the condition are to be compared.

InData provides a variety of operators for use in forming **if** statement conditions. They are intentionally similar to operators found in many programming languages. The InData comparison operators are listed in the following table.

OPERATOR	MEANING	EXAMPLES
<b>is empty, = ""</b>	Test whether a field is empty (contains no data).	<b>«if a is empty»</b>
<b>is not empty, &lt;&gt; ""</b>	Test whether a field is not empty (contains data).	<b>«if a is not empty»</b> <b>«if a»</b>

OPERATOR	MEANING	EXAMPLES
<b>is, =</b>	Test whether the first value is the same as the second value.	«if a = b» «if a = "Smith"»
<b>is not, &lt;&gt;, !=</b>	Test whether the first value differs from the second value.	«if a is not prev a» «if a <> "Smith"»
<b>&lt;</b>	Test whether the first value is less than the second value.	«if a < 10» «if a < "M"»
<b>&lt;=</b>	Test whether the first value is less than or equal to second.	«if a <= b» «if 20 <= b»
<b>&gt;</b>	Test whether the first value is greater than the second value.	«if a > 10» «if a > b»
<b>&gt;=</b>	Test whether the first value is greater than or equal to second.	«if a >= b» «if a >= 100»
<b>contains</b>	Test whether the first string contains a specified substring.	«if a contains "ex"» «if a contains b»
<b>is in</b>	Test whether the first string contained anywhere within the second string.	«if "ex" is in a» «if b is in a»
<b>is not in</b>	Test whether the first string is not present in the second.	«if "z" is not in a» «if b is not in a»

Note that **contains** and **is in** perform the same test; they simply include the substring and main string in the opposite order. For example, the following two tests are equivalent:

**"J" is in first**  
**first contains "J"**

Although they are included in the table, **is empty** and **is not empty** are not really separate operators, but rather are special cases of the **is** and **is not** operators. «**a is empty**» is exactly equivalent to «**a=""**» because **is** is equivalent to **=**, and **empty** is a built-in constant equivalent to the empty (null) character string.

Comparisons are performed in the following manner. Each comparison operator first attempts to compare the two items as numeric quantities; if it cannot do so (because one or the other is not a well-formed number), then the two items are compared on a character by character basis as character strings.

Comparisons of character strings are performed based on the strings' relative alphabetical order. Alphabetic case is taken into account only for character strings which are otherwise equal. For example, case is not taken into account when comparing **Turtle** and **dove**, but it is taken into account when comparing **Dove** and **dove**. See the section in Chapter 13 entitled "How String Comparisons are Performed" for details on InData's comparison conventions.

In comparison operations, literal character strings technically need only be enclosed in double quotation marks if they have another meaning which might be misinterpreted by InData. Here is an example where the literal character string must be surrounded by double quotation marks:

```
«fields first, last, title»
«if first is "last"» ...
```

With the quotation marks, the statement compares the value in the field **first** with the character string **last**; without the quotation marks, it would compare the value in the field **first** with the value in the field **last** if there is a field named **last**, and otherwise, it would compare it with the literal character string **last**.

Despite InData's flexibility, however, it is good practice to get into the habit of quoting literal character strings in prototype statements. In this way, any future modifications to the data file structure, or to InData, will not break your existing templates.

Either two sets of straight quotation marks or opening and closing quotation marks may be used to delimit character strings, as in these examples:

```
«if a = "something"»
«if a = "something"»
«if a = "She said "Hi there!" to him."»
«if a = "She said "Hi there!" to him."»
```

The final two examples illustrate the method for including quotation marks within quoted character strings.

The two types of quotation marks may not be intermixed as string delimiters, however. Thus, the following prototype statement is illegal:

```
«if a = "Oooopps! Do not do this"» Incorrect use of quotes
```

## Some Example Conditional Prototype Statements

The prototype below contains examples of many of these operators:

```
«fields last, first, acctnum, balance, last_dep, aux_acct»
«last», «first»»
Primary Account: «acctnum»»
«if aux_acct»Aux. Account: «aux_acct»»
«endif»«if last_dep > 0»Last Deposit: $«last_dep»»
«endif»«if balance < 0»*** Overdrawn ***»
«else if balance = 0»Zero Balance»
«else»Balance: $«balance»»
«endif»
«if acctnum contains "999"»Gold Circle Account»
«endif»»
```

The first **if** statement, in line 4, merely tests whether the **aux\_acct** field is empty or not; if it isn't, then its contents are inserted into the formatted record after a prefix. This **if** statement's corresponding **endif** appears on line 5 so that the paragraph mark following the **aux\_acct** field is also conditionally inserted.

Similarly, the literal text **Last Deposit: \$**, the contents of the **last\_dep** field, and a carriage return are all included only if the value in the **last\_dep** field is greater than zero, and the final line of a record will be **Gold Circle Account** whenever the string **999** appears anywhere within the **acctnum** field.

Here are some sample typeset records:

**Smith, John**  
 Primary Account: 123-87-2  
 Last Deposit: \$100  
 Balance: \$256.92

**Smith, Kevin**  
 Primary Account: 142-22-9  
 \*\*\* Overdrawn \*\*\*

**Smith, Larry**  
 Primary Account: 999-23-7  
 Aux. Account: 187-22-9  
 Last Deposit: \$2102.87  
 Balance: \$15934.85  
**Gold Circle Account**

### if...else Chains

The preceding prototype also contained a more complicated example of an **if** statement than we've seen so far:

```
... «if balance < 0»*** Overdrawn ***¶
«else if balance = 0»Zero Balance¶
«else»Balance: $«balance»¶
«endif¶
...
```

This **if** statement uses an **else if** statement as its second clause rather than just a simple **else**, forming a chain of three choices depending on the value in the **balance** field. The initial **if** tests whether the value in the **balance** field is less than zero. If so, the string **\*\*\* Overdrawn \*\*\*** is included in the record. Otherwise, it next tests whether it is equal to zero. If so, the string **Zero Balance** is included; if not, the final **else** clause is taken, and its accompanying string (**Balance: \$**) and the value in the **balance** field are included.

**If—else if—else—endif** chains of any length such as this one may be used to construct prototype statements where different actions take place depending on which of several possible values a field (or expression) has; this type of program-

ming statement is known as a **case construction**. One use of it is to include the first non-empty field in a series of fields. For example, the following prototype will include the first non-empty field (if any) among the first five in each formatted record:

```
«if a»«a»«else if b»«b»«else if c»«c»«else if d»«d»«else if e»«e»«endif»
```

As these two examples have illustrated, only one final **endif** statement is necessary for the entire **if—else if** chain.



A more complicated example of a case conditional structure is found in the Mail Merge document in the InData Samples/Mail Merge sub-folder. In this case, an **if—else if** chain is used to treat incoming data records very differently based upon the value found in some particular field. Here is the heart of the prototype (we've highlighted the conditional statements):

```
«if key=1» I am pleased to inform you that your work has been judged outstanding by all concerned. Congratulations on a job well done.«else if key=2» I am pleased to inform you that your work has been judged above average. All of us at Poole thank you for your efforts.«else if key=3» I regret to inform you that your work at the present time does not fully meet the level required for the position you hold.«else» Your work at this time has been found to be satisfactory.«endif»
```

```
At the present time, you will continue in your position as «itemb». «if key=1» However, as a result of your superior performance, you will be promoted one grade within that position, with the associated increase in salary in addition to the company-wide 1.5% cost of living increase.«else if key=2» In recognition of your performance, you will receive a 3% salary increase in addition to the company-wide cost of living increase of 1.5%.«else» This year, you will receive a salary increase of 1.5%, corresponding to the company-wide cost of living adjustment.«endif»
```

This prototype is part of a longer one which produces evaluation form letters for employees of the Poole Corporation. Based on the value in the field named **key**, radically different letters result. Values of **1** denote employees who are receiving the most favorable evaluation, values of **2** indicate a better than average performance review, values of **3** indicate a substandard evaluation, and all other values are interpreted as satisfactory. In the second paragraph of the prototype, only the **key** values **1** and **2** are treated in a special way.

The greeting line of each letter also uses a conditional statement to address recipients with **key** field values of **1** by their first name (probably a *faux pas*):

```
Dear «if key<=1» «title» «last»: «else» «first»,«endif»¶
```

Note that in this case the final paragraph mark is *not* conditionally imported, since we want a new paragraph to begin after the greeting in either case.

## Alternate Forms of the if Statement

There are other, shorter versions of the **if** statement keywords:

COMPONENT	ALTERNATE FORMS
<b>if</b>	<b>[</b>
<b>else</b>	<b> </b>
<b>endif</b>	<b>], fi, end if</b>

Thus, all of the following **if** statements are equivalent if **last** is a field name:

```
«if last is not empty» «last» «else» «first» «endif»
«[ last <> "" » «last» «|» «first» «]»
«[ last » «last» «|» «first» «]»
```

In addition, the keyword **then** may appear before the closing right chevron mark in the **if** statement if desired:

```
«if last is empty then» «first» «endif»
```

The **then** keyword performs no function, but some people feel that it makes **if** statements more readable. (There is no abbreviation for **then** since it's only present for readability.)

## Forming Compound Conditions

More complex conditions—boolean expressions—and **if** statements are possible than those we have considered so far. InData provides two logical connectives and a negation operator for constructing compound conditions:

- and** Joins two conditions into a compound (boolean) expression that is true only if both component conditions are true;
- or** Joins two conditions into a compound (boolean) expression that is true if either or both component conditions are true.
- not** Inverts the value of the following logical expression.

For example, the following **if** statement will place the contents of the **balance** field into the formatted data only when its value is greater than zero and less than or equal to 10,000:

```
«if balance > 0 and balance <= 10000» «balance» «endif»
```

The **not** operator reverses the logical sense of the expression that it modifies. For example, the following **if** statement inserts the contents of the **fzip** field when the contents of the **country** field is something other than **USA** or **Canada** and when the **zip** field is empty:



```
«[ not (country="USA" or country="Canada") and zip="" » «fzip» «]»
```

### Use Parentheses for Grouping

As was illustrated in the preceding example, in compound conditions involving more than two conditions, parentheses should be used to indicate the order of evaluation. For example, the following **if** statement will include the **acctnum** and **balance** fields in the formatted data (separated by a colon and a space) if the value in field **balance** is less than zero and the value in field **closed** is zero or if the value in **balance** is greater than zero and the value in **closed** is one:

```
«if (balance < 0 and closed = 0) or (balance > 0 and closed = 1)» «acctnum»: «balance» «endif»
```

You will get different results when you evaluate the expression grouped in this way than you would with other ways of grouping it. Always use parentheses to avoid ambiguity in complex conditions, particularly since InData groups things in non-standardly, in many cases.

### Arithmetic Operators

InData prototypes also support the following arithmetic operators within prototype expressions:

<b>+</b>	Addition
<b>-</b>	Subtraction and negation: for example, if the current contents of field <b>a</b> is 5, then <b>a+3</b> is 8 and <b>-a</b> is -5.
<b>*</b>	Multiplication
<b>/</b>	Division
<b>mod</b>	Modulus (remainder after division): for example, <b>23 mod 5</b> is 3.

Care needs to be taken with both the **/** and **mod** operators to avoid division by zero which will produce a fatal error.

### Nested if Statements

Nested **if** constructions are also possible in InData prototypes. Consider this example taken from the detective directory prototype found in the InData Samples folder:

```
«if app<>empty»Appointed: «app». «if other» Other positions held:
«other» «endif»
«endif»
```

This construction differs from the **if—else if—endif** form we looked at earlier, where the first **if** statement in the chain whose condition is true is the one that determines how the record gets formatted. In this case, the second, nested **if** statement is *entirely* dependent on the first one; if **app** is empty, the value in **other** will never be tested, and **other** will never be inserted into the formatted records. That is why there are two **endif** statements: one to close each **if** statement. The para-

graph mark comes between the two of them, making it conditional upon the outer if statement—and hence on the value in **app**—and independent of the value in **other**.

## Comparing with the Previous or Next Record

InData allows you to use the **if** statement to compare the contents of any field in the current record with any data from the previous or next data record. The **prev** keyword—which may be abbreviated as **prev**—preceding a field name indicates that the data is to be taken from the preceding record. Similarly, the **next** keyword indicates that the data is to be taken from the following record in the data stream.

For example, the following prototype prints a header record each time the **dept** field's value changes in a data file:

```
«if dept <> prev dept» «dept»¶
«endif» «last», «first»      «ext»¶
```

The following prototype places a rule and some additional space after the current record whenever the value in the **group\_code** field is about to change in the next record:

```
«if group_code <> next group_code¶
    «product_code»      $«price»¶
«else¶
    «product_code»      $«price»¶
«endif¶
```

For the first imported record, all fields in the previous record are empty. Thus, the conditions **prev dept is empty** and **dept is not prev dept** (assuming **dept** is always non-empty) will be true when processing the first record with the preceding prototype.

Similarly, at the end of the data, all fields in the next record are empty. Thus, the conditions **next dept is empty** and **dept is not next dept** (under the same assumption as above) will be true for the final record.

## Avoiding Unwanted Blank Lines and Empty Text Boxes

We've already considered methods for avoiding unwanted blank lines within the formatted records: making the paragraph mark following the conditionally imported data part of the condition rather than literal text, as in these examples:

```
«if a» «a»¶
«endif» «if b» «b»¶
«endif» «if c» «c»¶

«if a»¶
«a» «endif» «if b»¶
«b» «endif» «if c»¶
```

```
«endif¶
```

```
«c»«endif¶
```

In the left example, all three fields are conditionally imported, and each one that is imported is placed into its own paragraph which terminates after the data is imported.

The right example is somewhat different. In this case, all three fields are again conditionally imported and placed within their own paragraph. However, each preceding paragraph is terminated when a non-empty field is found. For example, if field **a** has data in it, the prototype will first output a paragraph mark, closing the *preceding* paragraph, and then the data in field **a**.

Thus, in the first approach, each field is responsible for terminating its own paragraph. In contrast, in the second case, each field is responsible for closing the previous paragraph before beginning its own. Which approach to use depends on the effects you want to achieve in your final document.

Suppose you wanted a single empty paragraph between imported records. Consider these two prototypes:

```
«if a»«a»¶
«endif»«if b»«b»¶
«endif»«if c»«c»¶
«endif»¶
```

```
«if a»«a»¶
«endif»«if b»«b»¶
«endif»«if c»«c»¶
«endif»«if a<>"" and b<>"" and c<>""»¶
«endif¶
```

The prototype on the left will always place a new paragraph into the document after each data record is imported, via the final paragraph mark following the **endif**. This means that all non-empty data records will be separated by blank paragraphs, and an entirely empty data record will also produce an (additional) blank line.

In contrast, the prototype on the right explicitly handles the case of an empty record, placing a blank paragraph only when no field has been imported for that record. In this case, consecutive empty records will not produce multiple empty paragraphs.

Next box and next column characters can also be imported conditionally. For example, compare these two prototypes:

```
«name»¶
«if a2»«a2»¶
«endif»«if a3»«a3»¶
«endif»«ci», «st» «zp»
```

```
«name»¶
«if a2»«a2»¶
«endif»«if a3»«a3»¶
«endif»«ci», «st» «zp»«if next name<>""»
«endif¶
```

In the left prototype, every record ends with a new box character. This will result in an empty box after the final imported record. In the right example, the next box character is placed only when the next record's **name** field contains data, sup-

pressing the unnecessary new box character (and a possible empty last page) after the final imported record.

Finally, here is an even more complicated prototype which handles the case of possible blank records within the incoming data:

```
«if name is empty» «next» «endif»
«name»¶
«if a2» «a2»¶
«endif» «if a3» «a3»¶
«endif» «ci», «st» «zp» «if next name<>""»
«endif»
```

This prototype assumes that an empty **name** field indicates a blank or invalid record that ought to be skipped. The **next** prototype statement—note that it is enclosed in chevron marks—tells InData to go on to the next data record, beginning processing at the start of the prototype. Thus, the next box character will be placed into the document only when a valid record is followed by another valid record. Be careful not to confuse the **next** statement with the **next** field modifier or the **next repeat** statement (to be discussed later in this chapter).

## Doing Something Every $n^{\text{th}}$ Record

Conditional statements may also be used to perform some operation to or after every  $n^{\text{th}}$  imported record. For example, the following prototype creates a simple features chart, shading every other line:

```
«if recordnumber(true) mod 2 = 1»
«a» » «[b="x"]»•«[b="n"]»n/a«[b="x"]» »
«[c="x"]»•«[c="q"]»?«[d="x"]»•«[d="n"]»n/a«[c="q"]»?
«else» «a» » «[b="x"]»•«[b="n"]»n/a«[b="x"]» »
«[c="x"]»•«[c="q"]»?«[d="x"]»•«[d="n"]»n/a«[c="q"]»?
«endif»
```

The basic importing statements are repeated twice within the prototype: once in a paragraph with a grey rule behind it and again in an unruled paragraph. The first field in each record is imported as is. The remaining three records are translated into symbols based on their contents.

Which section of the prototype is used to import a given data record is controlled by the **«if recordnumber(true) mod 2 = 1»** statement, where **mod** is the modulus operator (remainder after division) and the **recordnumber(true)** function returns the position within the import stream of the current record. Thus, the condition is true for odd numbered records and false for even numbered records.

Here is the resulting table, along with its preceding header line:

SPECIES	EXT. SKULL	TELEPATHS	WARRIORS
Centauri		•	•
Human		•	•
Mimbari	•	•	•
Narn			•
Vorlon	n/a	?	•

If you wanted to shade every fifth line, you would use the statement:

```
«if recordnumber(true) mod 5 = 1»
```

whose condition would be true for the first, sixth and eleventh records and so on. If you wanted to shade every fifth line starting with line five, then the condition to use is

```
«if recordnumber(true) mod 5 = 0»
```

If you wanted to shade every third line starting with line two, then the condition to use is:

```
«if recordnumber(true) mod 3 = 2»
```

If you wanted to shade every third line skipping the first eight lines, then the condition to use is:

```
«if recordnumber(true) mod 3 = 0 and recordnumber(true) > 8»
```

Finally, if you wanted to shade line five and every third line thereafter, then the condition to use is:

```
«if (recordnumber(true) ≥ 5 and ((recordnumber(true) - 5) mod 3 = 0)»
```

Adding extra space every n<sup>th</sup> line (or performing some other action) is handled in a similar manner, as in this example which places a blank line after every twentieth line:

```
«partnum» » «descr» » $«price»95¶
«if recordnumber(true) mod 20 = 0»¶
«endif¶
```

Methods for doing something every n<sup>th</sup> imported record when you are conditionally importing entire records are discussed in Chapter 10.

Finally, let's examine another version of the features table:

SPECIES	EXT. SKULL	TELEPATHS	WARRIORS
Centaurs		■	■
Human		■	■
Mimbari	■	■	■
Narn			■
Vorlon	N/A	?	■

Here is the prototype that created it:

SPECIES	EXT. SKULL	TELEPATHS	WARRIORS
..«a»» «if next a» «else» « »» «[b="x"»■« »» «if next a» «else» « »» «[c="x"»■« »» «[c="q"»? «if next a» «else» « »» «[d="x"»■« »» «[d="n"»N/A« »»			

It is similar to the one used before, although the symbols have changed. Notice that the vertical lines in the formatted table are created using vertical bar characters which have been vertically scaled and baseline shifted. Both the vertical rules and the data are positioned using center tab stops, and each paragraph has a horizontal rule underneath it. (We've increased the leading here to make the prototype easier to read.)

Here is this part of the prototype that created the vertical rules in a more readable form:

«if next a» «else» «|»»

The first bar is 10 point Helvetica type, vertically scaled by 270% and having a baseline shift setting of -2.5pt. This bar will be used in all but the final line of the table, where a somewhat shorter bar is needed to avoid crossing the final horizontal rule (220% vertical scale and a baseline shift of +1.5pt).

This technique requires a fair amount of trial and error to get things to look right, but the final result can be quite pleasing. You can use larger or smaller point sizes of Helvetica type to produce thicker or thinner vertical lines. In some cases, reducing the leading in the relevant paragraphs will also help in positioning the vertical bar characters to create vertical rules.

The Features document in the Samples folder uses some of the techniques we've discussed in this subsection for a significantly larger sample features chart. We recommend examining it if you plan to use InData in a similar way.

## Constructing and Using Loops within Prototypes

Suppose you wanted to make ten address labels for each record within a data file. You could repeat the prototype ten times, but loops provide another, easier way to perform this task.

The following prototype statements will create ten address labels for each record:

```
«repeat 10»  
«first» «last»  
«address»  
«city», «state» «zip»  
«end repeat»
```

The general form of a **repeat** loop is:

```
«repeat expression» loop body statements «end repeat»
```

When an integer expression is used as **repeat**'s argument, it may include field values, as in these examples:

«repeat b»...	<i>Field b's contents control # of times loop repeats.</i>
«repeat b-2»...	<i>Repeat loop b-2 times.</i>
«repeat b+c»...	<i>Add contents of fields b and c and loop that many times.</i>

For example, the following prototype repeats a given address label **count** times, where the paragraph consisting of «name» has a space-before setting of **15"** to force jumping to the next label frame:

```
«repeat count»  
«name»  
«address»  
«city», «state» «zip»  
«end repeat»
```

*Special space-before setting on this paragraph.*

This is an example of a loop that is (potentially) repeated a different number of times for each imported record.

Like **if** statements, **repeat** loops may be nested within prototypes. Each loop will require its own **end repeat** statement.

### Other Forms of the repeat Statement

Here are the various formats that a **repeat** statement can take:

```
«repeat integer-expression» loop body statements «end repeat»  
«repeat for integer-expression times» loop body statements «end repeat»
```

Loops using this form repeat the prototype statements in the loop body the specified number of times. The second form is a more verbose equivalent of the first one.

```
«repeat until condition» loop body statements «end repeat»
«repeat while condition» loop body statements «end repeat»
```

These forms of the **repeat** statement continue the loop until some logical condition is met. The **repeat until** form repeats the loop until the specified *condition* becomes true—in other words, as long as it remains false. In contrast, the **repeat while** form repeats the loop as long as the specified *condition* remains true—in other words, until it becomes false.

Here are examples of these types of **repeat** loops:

<pre>«repeat while dept="R&amp;D"¶ «last», «first» («degree»)¶ Ext. «extension»¶ «read» «end repeat¶</pre>	<pre>«repeat until balance&lt;0¶ «acctnum» is overdrawn!¶ Balance = \$«balance»¶ «read» «end repeat¶</pre>
--	--

The left loop processes records until the **dept** field holds something other than **R&D**; each subsequent record is retrieved by the **read** statement (discussed in detail in Chapter 10). The loop on the right processes records until the **balance** field ceases to hold a negative value (again using the **read** statement to retrieve records).

Note that if the initial condition is not met for these types of loops, then the loop is not processed even once. For example, if the **dept** field holds **Admin** when the loop on the left is encountered, then the loop will be skipped, although it may be encountered again when some later record is processed.

The following loop forms function like loops in high-level programming languages:

```
«repeat with variable = integer-expression1 to integer-expression2» ...
«repeat with variable = integer-expression1 down to integer-expression2» ...
```

These loops initially set the specified global InData variable to the result of *integer-expression1*, and increment or decrement it by one each time through the loop. The loop terminates when the variable passes the value in *integer-expression2* (exceeds it or falls below it).

For example, the left loop will run through five iterations, setting the global variable **ind** to 1, 2, 3, 4 and 5. The right loop will run through three iterations, setting the variable **ind** to 10, 9 and 8:

<pre>«repeat with ind=1 to 5¶</pre>	<pre>«repeat with ind=10 down to 8¶</pre>
-------------------------------------	---



Three important points need to be kept in mind when using these types of loops:

- ◆ The *variable* must be an InData variable and **must not** be a field name.
- ◆ The integer expressions in the **repeat** statement are evaluated only once, at the start of the loop. Consider this statement:

```
«repeat with ind=count1 to count2+1»
```

If **count1** and **count2** are variables, then whatever values they have when the **repeat** statement is executed are the ones that will be used to define the loop, regardless of how the variables' contents may change during the course of the loop.

- ◆ For ascending loops, the initial value, in *integer-expression1*, must be less than the value in *integer-expression2*, or the loop will be skipped. Similarly, for descending (**down to**) loops, *integer-expression1* must be greater than *integer-expression2*. When the two values are equal, then the loop is executed exactly once.

The specified variable retains the value it had during the last loop iteration after the loop terminates. If the loop is skipped, it retains its initial value (*integer-expression1*).

```
«repeat forever»
«repeat»
```

Both of these forms do what **repeat forever** implies, and you will have to use an **exit repeat** statement to break out of the loop (described in the next subsection). However, we recommend you use a **repeat n times** form instead, using a very large value for *n*, as InData will never relinquish control back to InDesign if you happen to have a bug in your prototype statements.

### Leaving a Loop Early

The **«exit repeat»** statement allows you to break out of a **repeat** loop early. For example, if you don't know ahead of time how many times a **repeat** loop should execute, you could use something like:

```
«repeat 1000000¶
prototype statements
«if balance > 100000» «exit repeat» «endif¶
prototype statements, probably including read
«end repeat»
```

When the value in the **balance** field is greater than 100,000, then the loop will terminate.

## Ending a Loop Iteration Early

The «**next repeat**» statement is used to skip the rest of the current **repeat** loop's body, going on to the next iteration. For example:

```
«repeat 1000000¶  
  «if degree<>"PhD"» «next repeat» «endif¶  
  Dr. «first» «last»¶  
  additional prototype statements  
  «if next dept="Admin"» «exit repeat» «endif¶  
«read» «end repeat¶
```

Note that a **next repeat** statement encountered outside of a **repeat** loop is treated as a **next** statement (discussed in Chapter 10), and an **exit repeat** statement encountered outside of a **repeat** loop is treated as an **exit** statement (discussed earlier in this chapter).

## Using Picture Frames in a Loop

You can use anchored picture frames inside repeat loops. The only restriction is that any **set** statement referencing a given picture box (e.g., «**set the filename of picture** *n*») must be at the same loop level as the anchored picture frame. (The “same loop level” includes being outside any repeat loops altogether, as in previous versions.)

There are no restrictions on the depth of loop nesting nor the number of anchored boxes inside each loop.

# 7

## Manipulating Incoming Data

This chapter discusses ways of manipulating data as it is imported: extracting parts of it, transforming it, converting it, and so on. It also covers general character string manipulation within InData

### Extracting Parts of Fields and Expressions

InData offers several methods of extracting part of the data in a field (or any string). The InData **character** operator may be used to extract substrings of field values. It has the following form:

**character** *n* [to *m*] of *expression*

where *n* and *m* are expressions for the starting and ending indices—offsets—of the characters to extract from the value in the specified expression, often just a field name. **Character** may be abbreviated as simply **char**, and the **to** *m* part of the operator is optional if you only want to extract a single character. Character indices start at 1 and go up to the length of the string; any index out of these bounds is quietly normalized to the nearest bound—e.g., 0 or -5 becomes 1, and index 10 of a 4-character string becomes 4. If *m* is less than *n*, then the resulting string is always empty.

For example, the following expression extracts the first character of the **last** field:

**char 1 of last**

The following expression extracts the third through the last characters from the **partnumber** field:

**character 3 to length(partnumber) of partnumber**

In a more complicated example, the following **if** statement compares the first character of the current and previous values of the **last** field:

**«if char 1 of last <> char 1 of prev last»**

Here are some more examples (assume that the field **title** holds **The Narnian Chronicles**):

EXPRESSION	RESULTING STRING
<b>char 1 to 4 of title</b>	The
<b>char 5 to 10 of title</b>	Narnia
<b>character 1 of title</b>	T
<b>char 2 to 2 of title</b>	h
<b>char 2 to 1 of title</b>	<i>empty string</i>
<b>character 100 of title</b>	<i>empty string</i>
<b>character 1 to length(title) of title</b>	The Narnian Chronicles

The following prototype uses **char** to reformat a phone number:

```
«first» «last» » «char 1 to 3 of phone» «char 4 to 6 of phone»•«char 7 to 10 of phone»¶
```

The phone field holds a 10-digit phone number without any formatting. The formatted records look like this:

```
James Kirk      510 555•1212
```

The following prototype reformats a price based on its number of digits:

```
«item» » «if length(price) <= 2» «price»¢ «else if length(price) >= 3» $ «char 1 of price» «char 2 to 3 of price» «else» $ «char 1 to 2 of price» «char 3 to 4 of price» «endif»¶
```

This prototype appends a cents sign if the data in the **price** field has only 1 or 2 digits. If it has 3 or 4 digits, the prototype puts a dollar sign in front of the first character, a period before the final two characters, and formats the final two characters in smaller, underlined, raised type.

Here are some sample formatted records:

```
Ink           $12.35
Pens          $1.23
Rubber Eraser 59¢
Parchment Ream $05.00
```

The prototype aligns all the differently formatted prices by using a right tab.

As the final example indicates, the previous prototype works only for prices of four or fewer digits. Here is an only slightly more complex version that works for any price:

```
«item» «if length(price) <= 2» «price»¢ «else» $ «char 1 to length(price)-2 of price» «char length(price)-1 to length(price) of price» «endif»¶
```

This prototype formats two digit prices with a cents sign following them. For all other prices, it outputs a dollar sign, followed by all but the last two digits of the price field, followed by a decimal point (period), followed by the final two digits, set in smaller, raised, underlined type. Here is how it formats the final price:

**Parchment Ream      \$105.00**



Let's consider a more complex prototype using the **character** statement. It formats a car parts price list. This document is included in the Samples folder as Car Parts:

```
«fields partno, note1, note2, list, discount»
«if char 1 of partno = "*"»
    «char 2 to 500 of partno»
«else»
    «char 1 to 3 of partno»-«char 4 to 20 of partno» » «note1» » «note2»
    «list» » «discount»
«endif»
```

The prototype uses **char** to format the part number and also to recognize and format the header records—distinguished by an initial asterisk—whenever they appear in the input.

Here are some sample formatted records, along with their fixed column heading line:

Part Number	Note 1	Note 2	List Price	Discount Price
<b>Transmission</b>				
400-102000	EF	3	74.50	<b>37.25</b>
400-102001	A	9	86.50	<b>43.25</b>
400-102002			60.00	<b>30.00</b>
400-102003	B	1	34.50	<b>17.25</b>
400-102008	KEB	9	30.00	<b>15.00</b>
400-102009	L	1	63.50	<b>31.75</b>
400-102010	JK		74.00	<b>37.00</b>
400-102011	FCJ		22.00	<b>11.00</b>
400-102012	G	6	41.50	<b>20.75</b>
<b>Alternator</b>				
400-102013		1	25.50	<b>12.75</b>
400-102014	D	4	108.50	<b>54.25</b>
400-102015	E	4	89.00	<b>44.50</b>
400-102016	A		93.00	<b>46.50</b>
400-102017	A	1	23.50	<b>11.75</b>

## Finding the Length of a Character String

The **length** function gives the total length (number of characters) of any field or expression. It has two equivalent forms:

**the length of** *expression*  
**length**(*expression*)

Which one to use depends only on your personal preference.

As the previous prototype indicates, the arguments to character specifying the starting and ending character indices need not be literal numbers, but can be any expression that evaluates to a number, including ones involving arithmetic operations.

As we noted in the previous chapter, InData supports the following arithmetic operations:

<b>+</b>	addition
<b>−</b>	subtraction
<b>*</b>	multiplication
<b>/</b>	division (with truncation)
<b>mod</b>	modulus (remainder after division)

These operations may be used for whole numbers (integers) only; InData does not support arithmetic involving non-integer numbers such as prices with fractional components. It does support comparisons involving non-integers, however.

Here is a more complex example using **length**. The following prototype formats each incoming record in one of two ways depending on the combined lengths of the **addr** and **city** fields:

```
«if length(addr)+length(city) < 25¶
«last», «first». «addr», «city» »¶                «phone»¶

«else
«last», «first» »¶                                «phone»¶
«addr», «city»¶

«endif¶
```

Here are some examples of the formatted records:

<b>Jones</b> , Tom. 12 Cherry Lane, London	<b>223-4433</b>
<b>Smith</b> , Phyllis 45638 Westminster Palace Drive, Morgantown	<b>243-5544</b>
<b>Wong</b> , Terrance. 954 Land St., Dover	<b>223-5544</b>

## Extracting Substrings

The **offset** function takes two arguments: a string to search for (the pattern) and a string expression in which to look for it:

```
offset(pattern, expression)
```

**Offset** returns an integer character position where the first occurrence of *pattern* begins in the specified *expression*, or zero if *pattern* is not found. Remember that character position numbering begins at 1.

For example, the following expression would evaluate to **5** if the field **last** held the value **Johnson**:

```
offset("son", last)
```

**Offset** is often used in conjunction with **char** to perform substring extractions whose starting and ending points are not known in advance. For example, the following prototype expression will insert all characters following the first hyphen in the **acct** field into the formatted data:

```
«char offset("-", acct)+1 to length(acct) of acct»
```

If there is no hyphen in the **acct** field, **offset** will return zero and the **char** operator will begin extracting at the first character (0+1), which is just what we want.

## String Concatenation

InData supports two string concatenation operators:

- &** Concatenate two character strings.
- &&** Concatenate two strings, adding an intervening space.

You can use these operators with literal character strings, fields, and general character expressions. Here are some examples (assume that the field **last** holds **Smith**, and **first** holds **Karen**):

EXAMPLE	RESULT
<b>first &amp; last</b>	KarenSmith
<b>first &amp;&amp; last</b>	Karen Smith
<b>first &amp; " Jones"</b>	Karen Jones
<b>first &amp;&amp; char 1 of last &amp; ". Jones"</b>	Karen S. Jones

Of course, within an actual prototype, these expressions would need to be enclosed in chevron marks. We'll see examples of these operators in action in later chapters of this manual.

## Including Literal Chevron Marks in a Prototype

The global named constants **guillemetleft** and **guillemetright** may be used to place literal « and » marks into formatted records, as in this example:

```
«fields homme, citation¶
«homme» a dit «guillemetleft»«citation» ...«guillemetright»
```

The typeset records appear as follows:

```
John F. Kennedy a dit «Ich bin ein Berliner ...»
Napoleon a dit «Able was I ere I saw Elba ...»
Josephine Baker a dit «J'ai deux amours ...»
```

## Extracting Words and Lines from Expressions

The **word** operator may be used to extract words from fields and general strings. Its format is very similar to **character**'s:

**word** *n* [**to** *m*] **of** *expression*

where *n* is the index of the first word you want (or an expression evaluating to a whole number), and *m* is the index of the final word you want (the **to** *m* part is optional if you only want to extract one word). Word indices begin at 1. Within the *expression*, words are normally separated by spaces.

Here are some examples (assume that the field **title** holds **Star Trek: The Next Generation**):

EXPRESSION	RESULTING STRING
<b>word 2 of title</b>	Trek:
<b>word 3 to 4 of title</b>	The Next
<b>word 8 of title</b>	<i>empty string</i>
<b>word 4 to 1000 of title</b>	Next Generation

As the final example illustrates, including a very large number as the final word to be extracted causes InData to select all remaining words. You can also use the **the number of words in** *expression* statement:

**word 4 to the number of words in title of title**

The **word** operator may be used to translate the numerical code fields commonly found in relational databases to descriptive strings. Here is an example:

```
raw data:      2    10   50
                3     1  150
                1     5   29
```



```
«fields prod_id, quantity, price»
«--the put statement defines a variable: see chapter 10»
«put "Chess Backgammon Checkers" into games»
«word prod_id of games» Sets:=> $«price * quantity».00»
```

```
output:      Backgammon Sets:    $100.00
             Checkers Sets:     $150.00
             Chess Sets:        $145.00
```

Within character string expressions, words are normally delimited by spaces, tabs, line feeds, carriage returns, and so on. Multiple delimiters in a row do not define additional words, but rather are carried along with the preceding word when multiple words are extracted:

EXPRESSION	RESULT
«a»	The big    bad wolf.
«word 2 to 3 of a»	big   bad

You can change the word delimiters using the **set worddelimiters** statement. For example, this prototype statement sets the word delimiter set to a comma and a semicolon:

EXPRESSION	RESULT
«a»	yes,no;maybe more words
«word 1 to 2 of a»	yes,no;maybe more
«set worddelimiters to " ;;"»	
«word 1 to 2 of a»	yes,no

Note that if you change the word delimiters in the middle of your prototype after having relied on their default values, you will need to set them back to the defaults afterward (e.g. at the end of the prototype) so that they will be set properly the next time through the prototype.

The **set wordcharacters** statement provides another way of customizing the **word** functionality. This statement specifies the characters that are considered part of words.

## Extracting Lines

The **line** operator similarly extracts lines from within an expression (where lines are separated by ASCII carriage return characters). It has this general format:

```
line n[to m] of expression
```

Here are some examples (assume that the field **song** holds the four lines of the song, “Row, Row, Row Your Boat”—we hope you know this one):

EXAMPLE	RESULT
<b>line 2 of song</b>	Gently down the stream,
<b>line 4 to 100 of song</b>	Life is but a dream.

You can also use the **the number of lines in** *expression* statement to determine the total number of lines in an expression:

**line 2 to the number of lines in song of song**

## Extracting Arbitrary Items

The **item** operator may be used to extract arbitrary items from fields and general strings. Its format is very similar to **word's**:

**item** *n* [**to** *m*] **of** *expression*

where *n* is the index of the first item you want (or an expression evaluating to a whole number), and *m* is the index of the final item you want (the **to** *m* part is optional if you only want to extract one item). Item indices begin at 1. Within the expression, items are normally separated by commas, although you can specify a different (single) delimiter with the **set itemdelimiter** statement.

Here are some examples:

EXAMPLE	RESULT
<b>«a»</b>	He was born, he assumed, on 5/2/97.
<b>«item 1 of a»</b>	He was born
<b>«item 2 to 3 of a»</b>	he assumed, on 5/2/97.
<b>«set itemdelimiter to "/"»</b>	
<b>«item 1 of a»</b>	He was born, he assumed, on 5
<b>«item 2 to 3 of a»</b>	2/97.

Note that consecutive items are separated by the item delimiter when they are output.

## Determining the Number of Chunks

The following prototype statements may be used to determine the number of characters, words, lines, or items within the specified expression:

<b>the number of chars in</b> <i>expression</i>	<i>Equivalent to</i> <b>length</b> ( <i>expression</i> ).
<b>the number of words in</b> <i>expression</i>	
<b>the number of lines in</b> <i>expression</i>	
<b>the number of items in</b> <i>expression</i>	

## Handling Repeating Fields

InData can handle repeating fields—fields which can hold more than one value in the same record—in data exported from FileMaker Pro and other databases supporting this feature, via InData’s **subfield** operator (abbreviable to **sfld**). The **subfield** operator extracts one subfield from a repeating field; for example, «**subfield 2 of k**» extracts the second subfield of field **k**, for example.

Consider the following completed invoice:

SOLD TO: Harry Jones Jones & Daughters, Scriveners 123 W. Main Street Freedom, WI 78987-3344 299-345-3564				
DATE MAY 22, 2088		NOTES	TERMS: NET THIRTY DAYS	
QUANTITY	DESCRIPTION		UNIT PRICE	TOTAL
3	Quill pens		2.50	7.50
4	Blotters		1.50	6.00
1	India Ink (5 oz. bottle)		15.99	15.99
5	Erasers		0.79	3.95

The structure of the invoice—the locations of the text frames, their header text and shading, and so on—are created on the document’s master page. The prototype is placed as usual in the text frames on page one of the document.

The address part of the invoice consists of normal prototype statements like the ones we’ve looked at so far; each line contains one or more field placeholders, and some lines include **if** statements for conditional importing. This section of the prototype consumes the first eight fields in each record, and it ends with a new frame character to force the next field into the main item list text frame (below the header line starting with QUANTITY).

The prototype for the item list extracts data from four repeating fields in the data. Here are the prototype statements in the item list area of the invoice:

```
«repeat with num = 1 to 40¶
«if subfield num of i is empty»«exit repeat¶
» «sfld num of i» » «sfld num of j» » «sfld num of k» » «sfld num of l»¶
«end repeat»
```

This prototype is based around a **repeat** loop indexed by the variable **num**. This loop allows up to 40 subfields to be processed for each repeating field. Our prototype processes four repeating fields in parallel. The third line extracts the successive subfields from fields **i** through **l** and places them into the invoice. The

fourth prototype line ends with a new frame character, forcing the remainder of the prototype into the total area of the invoice, where each record's final three fields are entered into the formatted invoice.

If desired, the subfield delimiter character may be defined via the **InData=>Preferences=>Data...** menu item. The default is set correctly for FileMaker Pro.

## String Conversion Functions

InData provides several functions for transforming character expressions:

FUNCTION	DESCRIPTION
<b>trim</b> ( <i>expr</i> )	Removes leading and trailing spaces in <i>expr</i> .
<b>trim</b> ( <i>expr1</i> , <i>expr2</i> )	Removes all characters found in <i>expr2</i> from the beginning and end of <i>expr1</i> .
<b>uppercase</b> ( <i>expr</i> )	Translates <i>expr</i> to uppercase.
<b>downcase</b> ( <i>expr</i> )	Translates <i>expr</i> to lowercase.
<b>wordcase</b> ( <i>expr</i> )	Capitalizes the first character of each word to uppercase (based upon current word delimiters), and converts all other characters to lowercase.
<b>dec2frac</b> ( <i>expr</i> )	Returns the whole number plus fraction equivalent (as a string) of its argument interpreted as a decimal number.
<b>numtochar</b> ( <i>integer-expr</i> )	Interprets an integer as an ASCII value and converts it to the corresponding ASCII character (as a single character string).
<b>chartonum</b> ( <i>char-expr</i> )	Converts a single character to its integer ASCII value.

We will now consider some examples (assume that **a** holds “ **It was a VERY Good year!** ” and **b** holds “ **Some text here.** ”):

EXAMPLE	RESULT
« <b>trim</b> (a)»	<b>It was a VERY Good year!</b>
« <b>trim</b> (a, " ! r l ")»	<b>t was a VERY Good yea</b>
« <b>trim</b> (a, " ! ")»	<b>It was a VERY GOOD year!</b>
« <b>trim</b> (b, " . ")»	<b>Some text here</b>
« <b>trim</b> (b, "Some text. ")»	<b>her</b>
« <b>uppercase</b> (a)»	<b>IT WAS A VERY GOOD YEAR!</b>
« <b>downcase</b> ( <b>trim</b> (a))»	<b>it was a very good year!</b>
« <b>wordcase</b> (word 1 to 4 of a)»	<b>It Was A Very</b>
« <b>dec2frac</b> ("1.5")»	<b>1 1/2</b>
« <b>dec2frac</b> ("500.125")»	<b>500 1/8</b>
« <b>dec2frac</b> ("400.3456")»	<b>400 34/100</b>
« <b>dec2frac</b> ("1 XYZ .375")»	<b>1</b>

Here is a prototype that sets the fractions resulting from **dec2frac()** in a more aesthetically pleasing way (we have broken the prototype into lines for readability, but there are not paragraph breaks anywhere within it):

```
«set itemdelimiter to "/"»
«put dec2frac(a) into frac»
«if the number of words in frac = 2»
«word 1 of frac» «item 1 of word 2 of frac»/«item 2 of word 2 of frac»
«else if frac contains "/"»
«item 1 of frac»/«item 2 of frac»
«else»
«frac»
«endif»
```

If **dec2frac(a)** results in 2 words, then there is both a whole number and a fraction; if it returns only one word, then we test for the presence of a slash to see if the result is a whole number or a fraction (having set the **itemdelimiter** to a slash at the beginning of the prototype).

When we find a fraction, we use the **item** operator to extract the numerator and denominator separately. We set both in smaller type, raising the baseline for the numerator, and we replace the slash with a virgule character from the PostScript Symbol font (Option-6 or Alt-0164).

Here are some sample formatted records:

```
1
1/2
5001/8
40034/100
```

We haven't bothered to kern the characters in the fraction. Note that if you want to kern a single-character denominator, you'll need to add a space before the closing chevron to avoid an InData error.

Here are some examples of **numtochar** and **chartonum**:

EXAMPLE	RESULT
«numtochar(65)»	<b>A</b>
«numtochar(999)»	<i>empty string</i>
«chartonum("B")»	<b>66</b>

The **chartonum** function can be used to determine whether a character is lowercase or not. The following prototype statement will determine whether the first character of the contents of field **a** is lowercase or not (with a true condition indicating a lowercase letter):

```
«if chartonum(char 1 of a) >= chartonum("a") and
chartonum(char 1 of a) <= chartonum("z")» ...
```

Note that non-letters (e.g. numbers) are considered non-lowercase by this test.

Here is a prototype which will test whether there are any non-lowercase letters within **a**, leaving the result in the variable **is\_lc** (the value **true** means **a** is entirely composed of lowercase letters):

```
«put true into is_lc»
«repeat with ind = 1 to length(a)»
«if not(chartonum(char ind of a) >= chartonum("a") and chartonum(char ind of a)
<= chartonum("z"))»
«put false into is_lc»
«endif»
«end repeat»
«if is_lc» ...
```

## Determining the Current Record or Page Number

We've already seen the **recordnumber** function in action. Here we discuss it formally and also introduce the **pageinfo** function.

The **recordnumber** function returns the record number of the current record within the data. It returns its absolute number if **recordnumber**'s argument is **false**, counting the first record in the data as record 1; it returns its position with respect to the number of processed records if **recordnumber**'s argument is **true**, defining as record 1 the first record imported after skipping records in accord with the settings in the **Record Range to Import** dialog.

For example, the following prototype statement uses **recordnumber** to place some text into the output every fifty records:

```
«if recordnumber(true) mod 50 = 0» some text»
«endif»
```

The **pageinfo** function returns information about the insertion point location in the document during importing. It has the following forms:

FORM	RESULT
<b>pageinfo(1)</b>	Absolute page number in document.
<b>pageinfo(2)</b>	Absolute spread number in document.
<b>pageinfo(3)</b>	Relative page number within the current section of the document.
<b>pageinfo(4)</b>	Page location within the current spread (left page=1, right page=2).
<b>pageinfo(5)</b>	Name of the master page applied at this point.

Page and spread numbering within the document and page numbering within each document section always begins at 1 (regardless of the **Page Numbering** settings in the **Page=>Section...** dialog).

For example, if you wanted to have a name be flush right on recto (right-hand) pages, and flush left on verso (left-hand) pages, you might use a prototype like this one:

```

«first» «last» «if pageinfo(1) mod 2 is 1»¶
«else»¶
«endif»¶

```

where the first paragraph is flush right, and the second is flush left; what happens is that either the first paragraph mark (and its right justification) takes effect, or the second (and its left justification), depending on whether the current page number is odd or even.

Note that using **pageinfo** at the start of a paragraph is error-prone, since the paragraph, without any text in it, may well fit at the bottom of one page, but once you add data, jump to the next. As you may surmise, it's *very tricky* to use **pageinfo** correctly and effectively, and data formatted using this function will obviously not readjust its format if you edit the imported data.

## Accessing Arbitrary Fields within Records

The function **fieldvalue** may be used to obtain the value of any desired field within the previous, current or next record. It has the general format:

```
fieldvalue (fnum [, which-rec])
```

where *fnum* is the number of the field whose contents are desired (field number begins at 1). The second argument to **fieldvalue** is optional. If included, *which-rec* specifies whether the value from the previous, current, or next record's *fnum*'th field is desired, corresponding to *which-rec* values of 1, 2, and 3, respectively.

**Fieldvalue** may be used to access the values of both named and unnamed fields within the data records, though you must declare the maximum number of fields you plan to use in your **fields** statement.

For example, **fieldvalue(1)** returns the value of the first field in the current record, **fieldvalue(25)** returns the value of the 25<sup>th</sup> field in the current record, and **fieldvalue(2,3)** returns the value of the second field in the next (upcoming) record. **Fieldvalue** returns an empty value if the requested field number is out of range or if a value other than 1, 2 or 3 is given for *which-rec*.

The function **fieldindex**(*field-name*) may be used to return the field index corresponding to a named field (as defined in the prototype's **fields** statement). For example, **fieldindex(name)** returns the field index for the **name** field, and **fieldvalue(fieldindex(name),1)** returns the value in the **name** field from the previous record.

Similarly, **fieldvalue(fieldindex(name) - 1 + idx)** returns the value of the *idx*'th field after the **name** field in the current record.

If the argument to **fieldindex** isn't a bona fide field name, then **fieldindex** returns the value 0.



# 8

## Importing and Formatting Pictures

InData can import variable picture elements along with text. Your input data must contain a picture filename (or pathname), as a field, for each picture you want to import, and the individual picture files must reside somewhere in the accessible file system. Essentially, InData is performing an automatic InDesign **Place...** command for each picture you import, with all that implies.

### Importing Pictures

Picture importing involves the following:

- ◆ including one or more anchored picture frames within the prototype;
- ◆ indicating the source of the filename for each picture frame.

It may also include these steps:

- ◆ specifying how pictures are to be placed into their picture frames, both globally and on a per-picture basis;
- ◆ specifying where InData is to look for picture files;
- ◆ specifying what to do if a requested picture file can't be found.

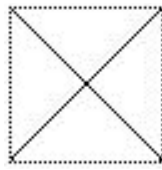
You can include an anchored picture frame in any InData prototype using the following procedure (which is the standard InDesign method for anchoring graphics within text):

- 1 Create a picture frame of the desired shape and size using one of the InDesign picture frame tools.
- 2 Select the picture frame with the selection tool, and then cut it to the clipboard.
- 3 Select the content tool, then place the insertion point in the prototype where you want the picture frame to go, and then choose **Paste** from the **Edit** menu .

The picture frame will now be within the prototype.

Here is a simple prototype containing a picture frame:

```
«fields last, first, picfile, title, addr, phone, fax, telex»
```



```
«set filename of picture 1 to picfile»
«first» «last»
«title»
«if addr»Business address: «addr»
«endif»«if phone»Telephone: «phone»
«endif»«if fax»Telefax: «fax»
«endif»«if telex»Telex: «telex»
«endif»
```

The field **picfile** holds the name of the picture file for each record. The prototype statement:

```
«set filename of picture 1 to picfile»
```

tells InData to use the contents of the **picfile** field in each record as the filename of the picture to put into the picture frame for that record. The picture number indicates which picture frame within the prototype the statement refers to (regardless of any conditional picture frame importing; see below). **set filename of picture** may be abbreviated as **set fn of pic**.

In InData 2.0, you can also import a page of a PDF file as a picture, using the **set pictureimportpage** prototype statement. For example, these statements import the page specified in the **pagenum** field of the PDF file specified in the **picfile** field:

```
«set filename of picture 1 to picfile»
«set pictureimportpage of picture 1 to pagenum»
«set pictureimportcrop of picture 1 to 2»
```

The final prototype statement sets the cropping for the imported page to the page itself. See chapter 13 for more information about these prototype statements.

### Templates and Imported Pictures

You can save documents with prototypes containing picture frames as InDesign templates. However, when you generate a new document from the template, you will need to save the document before pictures will import properly; generally, you would save the new document to the same folder where the picture files are located. Otherwise, InData will fail when trying to locate the pictures as records are

imported, *even when the template document resides in the same folder as the picture files* (and the error message in this case is quite confusing).

## Importing Pictures Conditionally

As it is now, the prototype expects every record to have a picture file. If this is not the case, then the picture frame may be conditionally imported based upon whether the field **picfile** is empty or not:

```
«fields last, first, picfile, title, addr, phone, fax, telex»
```



```
«if picfile»  
«set filename of picture 1 to picfile»«endif»
```

```
...
```

The complete version of the prototype (a tongue-in-cheek detectives directory) we examined earlier follows.

Each detective's entry has a "specialization profile" table, using a fixed set of icons to represent at a glance the detective's specialties. The InData template has 4 anchored graphics frames, each of whose contents are fixed, but only imported if the specialization profile raw data field contains the appropriate code. In order of their appearances in the frames (beginning at the left), these codes are: **m** for murder, **d** for divorce (and related affairs), **s** for spying/espionage, and **r** for ransom. The specialization code is the final field in each record (it has been added to the prototype's **fields** statement as **specprof**). For example, the specialization profile code field value **md** in a given record would call in the first and second icons.


If its corresponding code letter does not appear in the specialization profile field, then the filename of the picture file for each picture frame is never set, and so it remains empty. Note that the empty picture frames act as placeholders to keep the icons spaced in a fixed pattern.

```
«fields last, first, picfile, title, co, app, other, career, edu, natlty, addr, phone, fax,  
telex,specprof»  
«set pictureposition to aspectratiofit»  
«first» «last»  
«title»
```

```
«co»«if picfile<> empty»
```

```
»
```

```

«endif¶
«if picfile»«set filename of picture 1 to picfile»«endif¶
«if app»Appointed: «app».«if other» Other positions held: «other»
«endif»«endif»«if career»Career history: «career»¶
«endif»«if edu»Education: «edu»«if natlty»¶
Nationality: «natlty».«endif»¶
«endif»«if addr»Business address: «addr»¶
«endif»«if phone»Telephone: «phone»¶
«endif»«if fax»Telefax: «fax»¶
«endif»«if telex»Telex: «telex»¶
«endif¶
Areas of Specialization:  ¶

```

```

«if specprof contains "m"»«set fn of pic 2 to "murder.pict"»«endif¶
«if specprof contains "d"»«set fn of pic 3 to "divorce.pict"»«endif¶
«if specprof contains "s"»«set fn of pic 4 to "spy.pict"»«endif¶
«if specprof contains "r"»«set fn of pic 5 to "ransom.pict"»«endif¶

```



You may examine and experiment with this prototype if you like. It may be found in the Detectives folder in the InData Samples folder.

There are some sample typeset records:

## Susan M. Barnes

President  
Barnes & Daughters, P. I.



**Appointed:** 1980. **Other positions held:** Member, Private Investigators of America; Director, Boston Private Investigation Society; Senior Member, Dorothy L. Sayers Detection Club.

**Career history:** 10 years as co-owner of Hanratty Detection Service; 5 years as apprentice sleuth at Dominic & Associates.

**Education:** PhD Economics 1969, London School of Economics; MA 1965, Harvard School of Government; BA 1962, Radcliffe College.

**Nationality:** British.

**Business address:** 1011 Beacon Street, Boston, MA 01299

**Telephone:** 001 (617) 232-9912 **Telefax:** 001 (617) 232-9915 **Telex:** 822911

**Areas of Specialization:**

## Fenton C. Hardy

Chairman  
Hardy & Sons, P. I.



**Appointed:** 1930. **Other positions held:** Director, Hardy Investment Trust; Trustee, Hardy Share Ownership; Trustee, Hardy Rare Car Collection; Founder, Dorothy L. Sayers Detection Club.

**Career history:** Police Detective, New York City, 1920–1930. Founded private detection practice, 1930. Retired, 1965.

**Education:** BA 1918, Columbia University.

**Nationality:** American.

**Business address:** 120 Bayshore Lane, Bayshore, NY 10172.

**Telephone:** 001 (718) 606-4455 **Telefax:** 001 (718) 638-7291

**Areas of Specialization:**



## Setting Picture Frame Attributes

Keep in mind that the imported picture will take on *all* characteristics of the empty picture frame. These include:

- ◆ picture scaling;
- ◆ picture offset within the picture frame;
- ◆ angle of the picture within the picture frame;
- ◆ shearing (skew) of the picture within the picture frame;
- ◆ frame colors and shades;
- ◆ picture flipping (horizontal and/or vertical);
- ◆ clipping path (used for text wrapping);
- ◆ frame (stroke) around the picture frame.

In order to set the color and shade of imported pictures, along with any other image attributes, you will need to place a sample picture within the corresponding picture frame in the prototype and then set those attributes for that picture (the picture itself will be replaced upon import). All other attributes--i.e., frame attributes--may be set for an empty picture frame, and they will be inherited by the imported pictures. In fact, these values will override any that are present in the image itself (e.g., clipping paths).

You may also specify the placement and fit for all imported pictures within their picture frames using the **Default picture position** pop-up menu in the InData

**General Preferences** dialog (select **InData=>Preferences=>General...** from the InDesign menu).

The **InData General Preferences** dialog has the following options:

**Top left** Place the picture's upper left corner in the upper left corner of the picture frame (the InDesign default).

**Center** Center the picture within the picture frame.

**Center, Size to Fit**  
Scale the picture to fit the picture frame exactly.

**Center, Size to Fit, w/o Distortion**  
Scale the picture to fit the picture frame, maintaining its aspect ratio (original proportions), and then center it within the picture frame.

**Size Frame to Picture**  
Shrinks the anchored picture frame to fit its contents (once they are imported), obeying any scaling and margin picture frame properties (see chapter 5), and ignoring any offset picture frame properties.

**Size to Fit Horizontally, then Size Frame Vertically to Picture**  
Sizes the picture itself in its anchored picture frame to fit horizontally (i.e., to fill the frame in the "x" direction) once the picture is imported, sets the picture's y scale to match its x scale (as determined above by making it fit horizontally), and then shrinks the anchored picture frame itself vertically to make it fit the contained picture.

You can also set the picture handling for an individual picture frame with the **set pictureposition** statement (abbreviable to **set picpos**). We saw an example of this prototype statement in the Directory of Detectives. This prototype statement has the general form:

«**set pictureposition of picture *n* to *fit***»

where *n* is the index (number within the prototype) of the picture frame whose properties are to be set, and *fit* is a keyword specifying its placement and fit option, one of: **opleft**, **center**, **fit**, **aspectratiofit**, **framefit**, and **fitframev**.

## Prototype Statements for Setting Picture Frame Attributes

Many picture frame properties may also be set within the prototype, allowing you to control an anchored picture frame's height and width, the image's x and y offset (how the picture is shifted within the frame in either dimension), the image's

x and y scale (shrinking/expanding the picture within the frame in either dimension), and the image's x and y margin (how much extra or less space to give the picture in each dimension when using the two **Size Frame to Picture** options).

These properties are set using the following «set» statement formats:

«set [the] height of picture <i>n</i> to <i>measurement</i> »	<i>picture frame height</i>
«set [the] width of picture <i>n</i> to <i>measurement</i> »	<i>picture frame width</i>
«set [the] xoffset of picture <i>n</i> to <i>measurement</i> »	<i>image's horizontal offset</i>
«set [the] yoffset of picture <i>n</i> to <i>measurement</i> »	<i>image's vertical offset</i>
«set [the] xscale of picture <i>n</i> to <i>percentage</i> »	<i>horizontal image scaling</i>
«set [the] yscale of picture <i>n</i> to <i>percentage</i> »	<i>vertical image scaling</i>
«set [the] xmargin of picture <i>n</i> to <i>measurement</i> »	<i>horizontal image margin / trim</i>
«set [the] ymargin of picture <i>n</i> to <i>measurement</i> »	<i>vertical image margin / trim</i>

All *measurement* values default to points unless other units are explicitly specified (any measurement specification that you can type in the equivalent measurements palette edit field is supported), and *percentage* is a percentage value, with or without a trailing percent sign ("%"). If a given value is out of range, it is silently reset to the nearest minimum or maximum value (for example, you can't set the x or y scale to less than 10% or more than 1000%, and a specified value of 5% would result in 10% being used). If a picture is not imported into a given frame, because the picture import failed for some reason, only the height and width properties of the frame will be obeyed; all other properties you have set will be ignored.

For example, both of the following prototype statements set the height of picture frame 2 to one inch:

```
«set height of picture 2 to 72»
«set height of picture 2 to "1"»
```

Observe the two two curly quotes enclosing the 1 followed by a straight quote in the second statement.

Integer arithmetic expressions may be used for measurements. Thus, this prototype statement sets the width of picture frame 1 to 288 points (4 inches):

```
«set the width of picture 1 to 72*4»
```

The following prototype statement sets the scaling for the picture in the first picture frame to the value in a field **scale\_fac**:

```
«set xscale of picture 1 to scale_fac»
«set yscale of picture 1 to scale_fac»
```

The x and y margin settings are only used in the two **Size Frame to Picture** and **Size to Fit Horizontally, then Size Frame Vertically to Picture** picture positioning cases, and they specify how much margin to add (if positive) or subtract (if

negative) from the picture's natural width or height (respectively) on each edge before calculating the frame's width or height. These margins are in "document space," and thus are not scaled with the picture. If you don't specify these margins, they default to zero.

For example, if you wanted to cut off 2 points on each side of a picture horizontally before adjusting the frame to fit the picture, and after scaling the picture 50% in each dimension, but cut off nothing vertically, you'd use a sequence of prototype statements like the following:

```
«set xscale of pic 1 to 50»  
«set yscale of pic 1 to 50»  
«set xmargin of pic 1 to -2»
```

## Setting Default Directory Locations for Picture Files

By default, the picture filename in the data field must give the exact location of the picture file. If it is in the same folder as the current document, the filename alone is sufficient; otherwise, the complete path to the picture file must be specified.

However, you can give InData a list of locations to look in for pictures files, using the **set picturefolders** prototype statement; this statement requires a comma-separated list of folders as its argument. Folder names may be absolute (for example, Data:Pictures:, meaning the folder Pictures on disk Data), or relative (:Old Pictures:, referring to a folder named Old Pictures in the same folder as the current document). Note that on the Macintosh, all folder names use colons to separate one level from another, and also always end with a colon.

The principle is the same on under Windows. Folder names may be absolute (for example, C:\Data\Pictures\, meaning the directory \Data\Pictures on disk C:), or relative (Old\_Pix\, referring to a folder named Old\_Pix in the same directory and disk as the current document). Note that all folder names use backslashes to separate one directory level from another, and also always end with a backslash.

For example, if your document is named Spices, located in the folder named Catalog, and pictures it needs are located in that folder, the folder Old Pictures within that folder, and in the folder named Design on the disk named Themis, you would use the following **set picturefolders** statement:

```
«set picturefolders to ":Old Pictures:,Themis:Design:"»
```

Note that the **set picturefolders** statement must precede any **set filename** statement within the prototype which relies on it. Directories listed in this prototype statement may be regular directories or aliases (shortcuts) to them.

Multiple **set picturefolders** statements may appear in the prototype. If you use more than one of them, be sure that the initial path is always (re)set correctly at the start of the prototype.



It is also possible to specify the picture folder within the **set filename** statement, as in this example:

```
«set filename of picture 1 to ":Photos:" & pixfilename1 & ".TIF"»
«set filename of picture 2 to ":Drawings:" & pixfilename2 & ".EPS"»
```

These statements use the string concatenation operator **&** to add both a folder location and a standard extension to the picture filename that is imported via the **pixfilename***n* fields in each record.

## Handling Missing Picture Files

Normally, when InData can't find a requested picture file, data importing stops immediately, and InData displays an error message. You can change this behavior so that InData continues importing additional records anyway (and gives a warning about any missing pictures when finished). To do so, check the **Keep importing after missing pictures** checkbox in the InData **General Preferences** dialog.

You can also use the **fileexists(path)** function to determine if a given file exists.

## Specifying Precise Picture Locations on the Page

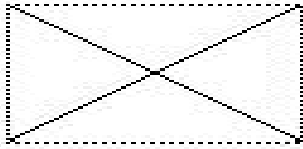
So far, we've considered only picture frames which flow along with the imported text. Sometimes, however, you will want to precisely position the picture frames on the page.



This may be accomplished in many cases by creating text frames in the desired locations on the document's master page and then using new frame characters to place text within them as records are imported. We'll consider an example taken from the Real Estate document in the Houses folder of the Samples folder.

The master page for this document has four text frames on it, placed in an artsy arrangement. The prototype will place one picture and its accompanying descriptive text into each text frame on each page:

Here is the prototype (we've shrunk the picture frame considerably):

```
«fields picfile,description,price¶
«set picpos to aspectratiofit¶
¶
«set filename of picture 1 to picfile¶
«put styled description»¶
$«price» «if next picfile is not empty»
«endif»
```

Here is the general layout of a completed page (examine the sample file itself for a closer look):

## Heritage Properties



Enjoy a wonderful view of Washington, D.C., from your front porch. This grand residence is imbued with history, and steeped in the tradition of American government. Buy now before taxes go up (thanks to this place)!

\$750,000,000



Get away from crazed Manhattan to the rest and peace of your own little oasis in *New York Harbor*. Property includes a well-known landmark (perpetual care provided), as well as free ferry service to and from Battery Park and Jersey City.

\$120,000,000



Gaze over the lovely water park from your free-standing conservatory (pictured). Close to downtown *San Francisco*, ideal for the harried commuter. *Won't last at this price!*

\$560,000,000



"If you lived here, you'd be home before you even started your commute!" Central *Manhattan* location, ideal for both up- and down-town commuting. No reasonable offer refused.

\$3,250,000,000

# 9

## Controlling Document Layout

InData can produce running headers and footers which change from page to page, based on the contents of the imported data. For example, a telephone directory might have the last names of the first and last people on each spread in its top outside corners, or a product list might have the product type in the page header.

There are three basic steps to creating a running header or footer:

- ◆ Placing a **mark** in the prototype—producing what we'll call **marked text** at data import time—which means designating an expression (usually a field) as the source of the running header/footer text.
- ◆ Adding a **mark reference** to a header or footer text frame on the document's master page, which serves as a placeholder for the final header or footer text.
- ◆ Importing the data as usual.

InData automatically computes the running headers and footers after importing all records by updating all mark references on all pages affected by the import operation.

### Creating a Mark in the Prototype

The **put marked** statement is used to tell InData to insert the contents of a field into each imported record and to give it a name by which it can be referred to in running headers/footers. For example, the following prototype statement inserts the **dept** field into the prototype, marking it as **A**:

```
«put dept marked "A"»
```

You may wonder why we have to assign a new name to the **dept** field when it already has one. The **marked** keyword is actually quite flexible, and it may be used to mark not only field names but any expression: **char 1 of lastname**, for example. In the latter case, a name needs to be assigned to the resulting text in order to refer to it from a header or footer.

All InData mark names consist of a single letter, and are case insensitive. The mark name can be a literal character string (in double quotation marks) or an expression. If the mark name is longer than one character in length, only the first character will be used.

The **hidden** keyword may be included after **put** to create a mark for text which you don't want to be visible in the formatted records. For example, the following **put** statement inserts a last name, prefixed with an invisible mark consisting of the first letter of the last name. A header or footer referring to mark **L** will pick up just the first character of the last name.

```
«put char 1 of last hidden marked "L"»«last»¶
```

One caveat is that **«put ... hidden ...»** actually uses a zero-width discretionary hyphen as the marked text placeholder, which will disable automatic (but not manual) hyphenation for any immediately-following word.

In fact, a more efficient method to achieve this same effect is:

```
«put char 1 of last marked "L"»«char 2 to length(last) of last»
```

## Adding a Mark Reference to the Master Pages

Once you've created a mark in your prototype, you must next define a mark reference to that mark before importing any data. A mark reference serves as a placeholder for the contents of a mark. It is created by entering some literal text into a header or footer frame on a document's master page, selecting it, and then designating it as a mark reference by using the InData **Make Header/Footer** dialog.

Note that the text frames into which the mark references are placed on the master page may not be linked to any other text frame (including each other).

The placeholder text used as a mark reference in running headers is simply any literal text that you choose to use. It requires no special chevron marks. It's a good idea to choose text that will remind you of what will replace it when data is imported. For example, you might enter the text **Dept** into a header to be used as a mark reference for the mark **A** we defined earlier.

Mark reference text can coexist with other text within a header or footer. For example, you may also include literal text within the header/footer, and you may include more than one mark reference within the same header or footer.

Next, apply any character and paragraph attributes and style sheets to the mark reference text. Be sure to do so *before* designating the text as a mark reference.

To designate text as a mark reference, select it with the content tool, and then select **Make Header/Footer...** from the InData menu.

This option brings up the **Make Header/Footer** dialog.



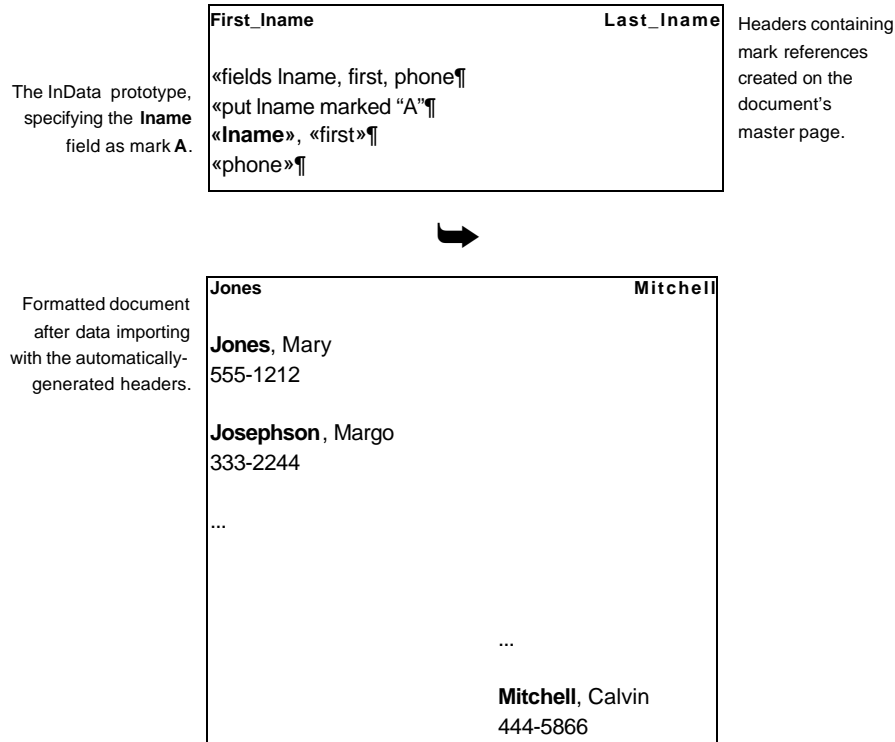
This dialog allows you to associate the first or last occurrence of a specified mark with the header or footer text you have highlighted, transforming the latter into a mark reference. Choosing **first** means that the first instance of the marked imported text on each page (or spread, if **spread** is checked) will be substituted in the placeholder's location on the page. Similarly, turning on the **last** radio button means that the final instance of the marked imported text on each page (or spread) will go into the page's header (or footer).

The **Limit initial search for the first occurrence of marked text** checkbox allows you to specify a range of lines in which the first marked text must be found in order for it to be used in the header or footer. You specify the number of lines in the text frame preceding **line(s)**. By default, marked text found anywhere on the page will be used.

The **Reference marks from previous pages** checkframe controls what happens if no marked text is found on a particular document page. If it is checked, then any marked text from the most recent previous page will be used. In this case, you can optionally append a string—for example, **continued**—to the text by entering it into the field following **appending**. Note that XPress Tags are supported in this field.

After importing data into the document, InData will automatically create the running headers and footers for each page, based on your mark references in the header and footer frames and any associated marked data in the formatted text on the page.

The following diagrams illustrate the header/footer creation process:



## Updating Existing Headers and Footers

Although running headers and footers will be created automatically when the data is imported, they are not updated automatically if you make subsequent changes to the formatted records. However, you can always instruct InData to update them at any time.

For example, suppose that you changed the definition of a paragraph style used in the formatted data, editing the space below or space above setting. This would cause QuarkXPress to reflow the formatted text, and possibly invalidate the InData-created running headers and footers (given that pages will now break at different points). To re-create the headers, you would place the insertion point in the story containing the imported data, and then choose **Update Headers/Footers...** from the **InData** menu; InData would then go through all the pages touched by the imported data, and synchronize all of the headers and footers with respect to the new data layout.

It's also possible to edit individual elements of (non-hidden) marked text, if you find errors, though it's slightly tricky. For example, suppose that you were using the **dept** field in a running header and you noticed a typographical error in the

first record on a given page. Then, this error would also appear in the page's running header.

One way to correct this error would be to return to the original database application, correct the error, re-export the data to a file for use by InData, and finally re-import that data.

While you probably would want to correct the error in the original database at some point, you could also directly correct the error in the imported record, and then use InData's ability to update the document's generated headers and footers. Note that you do not correct the error by editing the header/footer directly.

Only very minor changes—such as correcting typographical errors—may be made to the imported marked data and still be correctly updated in the document's running headers and footers. In particular, replacing the entire marked field contents will not work as desired, unless you take special precautions.

InData places hidden characters around the imported text; it searches for these hidden characters when creating or updating running headers and footers. If this mark is deleted—as it will be if you select the entire marked text and then type over it—then the mark for that data is forever lost. If you want to replace the entire contents of the marked text, then the only safe way to do so is to place the cursor somewhere in the middle of the string, add in the new text, and then carefully delete the old text from the “middle out”: in other words, use delete forward for characters to the right of the new text, and the delete key for characters to the left of the new text.

## Changing the Headers and Footers Themselves

It is also a bit tricky to change the format or content of the headers and footers themselves once you have imported data. Here are the steps for doing so:

- 1 Delete the existing mark reference text that you want to change. It is often a good idea to delete and recreate the entire header or footer.
- 2 Create and style new text for the header or footer.
- 3 Select the mark reference text, and designate it as such by selecting **InData=>Make Header/Footer...** from the menu.
- 4 Return to the normal document pages and import additional data (if necessary). Then instruct InData to update the headers and footers (**InData=>Update Headers/Footers...**).

## Applying Master Pages within a Prototype

InData provides two statements for applying a master page (or spread) to a document page or spread as data is imported:

```
«set master of this item to pagename
«set firstmaster of this item to pagename
```

where *item* is either **page** or **spread**, depending on which of them you wish to apply the master page/spread to, and *pagename* is the name of the master page/spread that you want to apply.

Master names may be specified by giving only the descriptive name that follows their letter designation or their complete name (including the letter prefix). For example, you could refer to master **B** as either **Chapter Start** or **B-Chapter Start**.

For example, the following prototype statement would set the master page of the current document page to master page **B** when it was encountered during data importing:

```
«set master of this page to "Chapter Start"
```

The **set master** and **set firstmaster** statements differ in how multiple occurrences of within the same page (or spread) are treated. For **set master**, the *final* instance of the statement is the one that will be used and accordingly determine which master page is applied. For **set firstmaster**, the *first* statement encountered on a page has precedence, and all subsequent **set firstmaster** statements are ignored.

We do not recommend mixing the two types of master page application statements within a prototype. However, should you do so, note that **set master** has precedence over **set firstmaster**.

Here is a more complex prototype using this feature:

```
«if fundname<>prev fundname» ⌵
«endif¶
«if fundage= 0» «set master of this page to "New Fund"¶
«else if fundage<5» «set master of this page to "Young Fund"¶
«else» «set master of this page to "Normal Fund"» «endif¶
```

This prototype formats a catalog of mutual fund offerings. Each mutual fund's entry begins on a new page (forced by the new frame character). The page can have three different layouts, depending on how long the fund has been in existence (and hence how much retrospective performance data there is to present). If the **fundage** field is non-positive, then the master page **New Fund** is applied to the current page. If this field is greater than zero but less than 5, then the master page **Young Fund** is applied to the current page. Finally, if the value in **fundage** is greater than 5, then the master page **Normal Fund** is applied.



Care needs to be taken when applying master pages to the current spread rather than the current page so that the desired results are accomplished. For example, if a **set master of this spread** statement is encountered on a right-hand (odd) page, then it will have the effect of potentially changing the layout of the *previous* document page, which may not always be what you intend.

Applying master pages with these prototype statements allows you to achieve a variety of effects, including the following:

- ◆ Applying different page layouts to different types of records.
- ◆ Using a special page layout for the first record within a group of records (for example, when the type of item changes within a catalog).
- ◆ Making headers/footers appear or change when the value in some specific field changes (or takes on some specific value).

These master page application features may be used in conjunction with the automatically generated header/footer capabilities we considered earlier in this chapter.



# 10

## Advanced Prototypes

This chapter discusses advanced features of InData prototypes. The InData prototype language includes the ability to define variables and solicit user input during the import process. We'll look at these and other advanced prototype statements and capabilities in this chapter.

### Record Input Control Statements

This section will formally introduce three prototype statements used to control how records are read relative to prototype processing: **next**, **read** and **exit**.

The **read** statement is used to read in the next record in the data file immediately and apply the remainder of the prototype to it. The record being processed before the **read** statement is encountered then becomes the previous statement for the purposes of the **previous** keyword.

The **read** statement is useful when repeating all or part of a record within a prototype; it tells InData when to go on to the next record of data. (Note that there is an implicit **read** statement at the start of every prototype.)

The **next** statement is used to skip the rest of the prototype for the current record and begin again at the beginning with the next record in the data file.

The **exit** statement is used to stop processing immediately, skipping all further records and any remaining prototype statements. InData returns control to InDesign whenever it encounters an **exit** statement.

The following prototype illustrates the use of the **next** and **exit** statements. It might be used to create a company directory:

```
«fields last, first, room, ext»«if last >= "M"»«exit»«endif»  
«last», «first»«if room < 200»  
«next»«else» «ext»  
Room «room»  
«endif»
```

Formatted records look like this:

Crashaw, Richard  
Room 515

272

The **next** command causes the **ext** and **room** fields' formatting to be skipped if the value in the **room** field is less than 200—if the room number is in the 100's, perhaps used for temporary offices.

If the **last** field's value is greater than or equal to the string "M"—in other words, if it is alphabetically at or after "M," which includes any string beginning with a M or a later letter in the ASCII collating sequence—then processing ceases (this presupposes that the data file is sorted by the **last** field). A technique such as this is useful for processing a data file in two passes, once with the comparison set to "greater than," and once with it set for "less than or equal" whatever is used as the cutoff point.

Here is a more complex example using **read** and **next**. This prototype formats a simple status report for credit card accounts. Some of the account numbers in its data refer to the same real account (for example, a husband and wife both having cards but sharing one account). Such accounts are set up to differ in their least significant account digit, so they will always follow one another in data files sorted by account number, and the primary account will always come first.

Another field in the data file—named **secondary** in the prototype—is a flag indicating whether a main account has a second card on it or not. This prototype checks the **secondary** field, and if it is set to 1 (on), it processes its record in the data file as well:

```
«fields acct,balfwd,lastpay,newpur,int,cadv,newbal,secondary»
Acct.: «acct» » Balance Forward: » «balfwd»
» Last Payment: » «lastpay»
» Interest: » «int»
» New Purchases: » «newpur»
» Cash Advance Fees: » «cadv»
» NEW BALANCE: » «newbal»
«if secondary<>1» «next» «else» «read»
2ndary Acct.: «acct» » Balance Forward: » «balfwd»
» Last Payment: » «lastpay»
» Interest: » «int»
» New Purchases: » «newpur»
» Cash Advance Fees: » «cadv»
» NEW BALANCE: » «newbal»
«endif»
```



A more complex example of **read** is found in the Features document in the InData Samples folder. The prototype processes two data records each time it is used, shading one imported record and leaving the other one unshaded, by performing a **read** operation to retrieve the second record. Here is the prototype:

«a» «[b="x"»•«[b="n"»n/a«[c="x"»•«[c="n"»n/a«[d="x"»•«[d="n"»n/a«[e="x"»•«[e="n"»n/a«[f="x"»•«[f="n"»n/a«[g="x"»•«[g="n"»n/a«[h="x"»•«[h="n"»n/a«[i="x"»•«[i="n"»n/a«[j="x"»•«[j="n"»n/a«[k="x"»•«[k="n"»n/a«[l="x"»•«[l="n"»n/a«[m="x"»•«[m="n"»n/a«[n="x"»•«[n="n"»n/a«[o="x"»•«[o="n"»n/a«[p="x"»•«[p="n"»n/a«[read]a«[b="x"»•«[b="n"»n/a«[c="x"»•«[c="n"»n/a«[d="x"»•«[d="n"»n/a«[e="x"»•«[e="n"»n/a«[f="x"»•«[f="n"»n/a«[g="x"»•«[g="n"»n/a«[h="x"»•«[h="n"»n/a«[i="x"»•«[i="n"»n/a«[j="x"»•«[j="n"»n/a«[k="x"»•«[k="n"»n/a«[l="x"»•«[l="n"»n/a«[m="x"»•«[m="n"»n/a«[n="x"»•«[n="n"»n/a«[o="x"»•«[o="n"»n/a«[p="x"»•«[p="n"»n/a«

Import records into the sample document in order to view some sample formatted records. Note that this can also be accomplished using the **recordnumber(true) mod 2 = 1** construct.

(Note that Microsoft Word’s mail merge facility also includes a “*next*” command. It has the meaning of InData’s **read** command. Don’t confuse them.)

## Setting Variables

InData supports user-defined variables. These variables may be used as constants (i.e., statically) or be updated throughout the import process.

For example, the following statement defines a variable **myname** and sets its value to the indicated string via the **put** statement:

**«put "Cassandra Temple" into myname»**

Once set, variable values are referenced in the same way as fields; enclosing a variable name in « and » marks causes its value to be placed into the prototype: «**myname**» would place **Cassandra Temple** into the formatted records, after the **put** statement above.

All variables start off empty; therefore, testing whether a variable is empty is one way to determine if it needs to be initialized. For example, this statement sets the value of **counter** to 1 if it is not defined; otherwise, it increments it:

```
«if counter=empty¶
«put 1 into counter¶
«else¶
«put counter+1 into counter¶
«endif¶
```

A variable like this one might be used to perform some action every so many records during an import process, where not every processed record is inserted into the document (otherwise, using **recordnumber** is much easier way of doing

this). For example, the following prototype creates one line in the formatted data for each record where the **part\_num** field is less than 100,000, placing a ruled line after every 25 formatted records:

```
...
«if part_num < 100000»«next»«endif»
«part_num»» «descr»» «unit»» «disc»           We've added a line break here
«if counter=empty»«put 1 into counter»          for readability.
«else»«put counter+1 into counter»«endif»
«if counter = 25»«put 0 into counter»»

«else»»                                         End the normal paragraph
«endif»
```

The first statement skips the rest of the current record if the **part\_num** field's value is too high. The second line inserts and formats four data fields. The third and fourth lines increment the value of **counter**, setting it to 1 if this is the very first record chosen for importing, and adding 1 to its current value in all other cases. Note that line three is really part of the second line, but we've inserted a line break in this manual for clarity. The final **if** statement (lines 6-8) determines whether **counter** has reached 25 yet. When it does, it places a ruled line (via the literal paragraph mark at the end of line 6) into the InDesign document, and resets **counter**'s value to 0 to start counting over again. Otherwise, it simply adds a normal paragraph mark to the end of the current line.

You can use similar techniques to produce complex patterns of ruling and shading within the formatted data.

## Manually Wrapping Text Columns by Words

Here is a more complex prototype which makes extensive use of InData prototype variables and **repeat** loops. Because of its length, we've included our comments within the prototype and we've formatted them for easier reading:

```
«—set up column widths (# characters) »
«put 25 into col1width»
«put 32 into col2width»
«put 15 into col3width»
«—copy fields to variables »
«put a into col1»
«put b into col2»
«put c into col3»
«repeat 50 —loop until all columns are exhausted, but »
« —use 50 as a fail-safe in case we have a bug :-» »
«—this is the loop for column 1 »
«put the number of words in col1 into nwcol1»
«repeat —fit as many words from col1 on this line as we can »
«if nwcol1 <= 1 or length(word 1 to nwcol1 of col1) <= col1width»
«exit repeat»
«endif»
```

```

«put nwcol1 - 1 into nwcol1  —not all the words fit: step back 1 word & check again.
¶
«end repeat¶
«—this is the loop for column 2¶
«put the number of words in col2 into nwcol2¶
«repeat —fit as many words from col2 on this line as we can ¶
«if nwcol2 <= 1 or length(word 1 to nwcol2 of col2) <= col2width¶
«exit repeat¶
«endif¶
«put nwcol2 - 1 into nwcol2  —not all the words fit ¶
«end repeat¶
«—and this loop is for column 3 ¶
«put the number of words in col3 into nwcol3¶
«repeat —fit as many words from col3 on this line as we can ¶
«if nwcol3 <= 1 or length(word 1 to nwcol3 of col3) <= col3width¶
«exit repeat¶
«endif¶
«put nwcol3 - 1 into nwcol3  —not all the words fit ¶
«end repeat¶
«—here is the line that actually outputs text: ¶
«word 1 to nwcol1 of col1» » «word 1 to nwcol2 of col2» » «word 1 to
nwcol3 of col3¶
«—remove the text we just wrote out from the three variables ¶
«put word nwcol1+1 to (the number of words in col1) of col1 into col1¶
«put word nwcol2+1 to (the number of words in col2) of col2 into col2¶
«put word nwcol3+1 to (the number of words in col3) of col3 into col3¶
«—test whether there is any text left in any of the column variables¶
«if length(col1) = 0 and length(col2) = 0 and length(col3)=0¶
«if we're done, output paragraph mark (with space after) & exit¶
¶

«exit repeat¶
«else —we're not done, so just output a Shift-Return to end the current line¶
_
«endif¶
«end repeat¶

```

Here are some sample formatted records:

#### COLUMN 1

There is a whole lotta  
lotta text going on and  
on and on and on.

Now is the time for all  
of the good women to come  
to the aid of themselves  
and their children.

#### COLUMN 2

One quick, brown fox jumping  
over the lazy dogs is as good as  
another.

Why is all this text so very very  
silly silly???

#### COLUMN 3

She sells sea  
shells on the  
leeward side of  
the seashore.

How many chuck  
steaks would a  
wood cutter's  
daughter eat if  
she were hungry?



You can examine this prototype in more detail by studying the Column Wrapping document in the InData Samples folder.

## Implementing Fixed-Width Fields

The **trim** function can be used in conjunction with character extraction and prototype variables to handle fixed-field input. To do so, first disable all **field** and **sub-field** delimiters and the **quote** character, and select an appropriate record terminator (if yours is non-standard) using the **Data Preferences** dialog (or the **Data...** button in the InData control panel). Disabling these fields is accomplished by removing all characters from the corresponding dialog field.

Then use a prototype like this one:

```
«—records contain the fixed fields name(25), addr(55), phone(20)¶
«fields inputLine¶
«—store fields into variables¶
«put trim(char 1 to 25 of inputLine) into name¶
«put trim(char 26 to 80 of inputLine) into addr¶
«put trim(char 81 to 100 of inputLine) into phone¶
«—output the data; this is a simple example: ¶
Name: «name»¶
Address: «addr»¶
Phone: «phone»¶
```

## Testing the Data Type of Expressions

InData includes the HyperTalk type-comparison operator **is a[n] type**, where *type* is **integer**, **number** or **logical**. This operator results in **true** if and only if its left operand can be legally construed as an element of its right operand type (otherwise, **false**, and no error results in any case). Here are some examples:

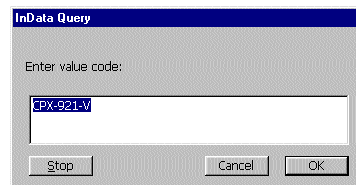
EXAMPLE	RESULT
«1 is an integer»	true
«1 is a number»	true
«1.1 is an integer»	false
«1.2 is a number»	true
«1.2 is a logical»	false
«true is a logical»	true
«false is a logical»	true
«true is an integer»	false
«“a” is an integer»	false
«“a” is a number»	false
«“a” is a logical»	false



## Soliciting Input at Import Time

The **ask** prototype statement may be used to obtain input during the import process. The **ask** statement must be followed by an expression which will be displayed to the user in a dialog. The question may optionally be followed by the keyword **with** and another expression. If the **with** clause is included, its expression will be used to pre-fill in the answer text in the dialog. Here is an example:

```
«if value=""»«ask "Enter value code:" with "CPX-921-V"»«endif»
```



The illustration shows a sample **ask** statement and the dialog it produces. The value returned by **ask** is placed into the built-in variable **it**; **it** may be used in prototype statements like any global variable and be inserted into the output records by enclosing it within chevron marks:

```
«ask ... »«if it<>""»«it»¶
«endif¶
```

Note that the expressions **«prev it»** and **«next it»** are not meaningful, since **it** is not a field.

### Asking More than One Question

If you want to issue multiple **ask** statements while processing a single record, then you may have to save the value of **it** in a variable after each one if you cannot use it immediately. For example:

```
«if price=""»«ask "Enter the price:"»«put it into newprice»«endif
«if quantity=""»«ask "How many?" with "1"»«put it into quan»«endif
```

### Asking Only Once

Sometimes you only want to issue a prompt a single time. For example, if you want to ask the user something prior to processing the incoming records, use a prototype like this one, which issues a prompt if the **filename** field in the first data record is empty:

```
«if recordnumber(true)=1 and filename is empty»«ask ...»«endif
import the data
```

If you want to ask the user something only once, but do not know in advance where within the imported records the prompt will be needed, use a prototype like this one:

```
«if recordnumber(true)=1»«put false into asked»«]»
«if a="" and not asked»«ask "Tell me:" with "yes"»
«put true into asked»«]»
«if a»«a»«|»«it»«]» «b»«]»
```

This prototype uses the variable **asked** to keep track of whether it has issued the prompt yet. Subsequent records with empty first fields will import the same response that the user gives when the first empty **a** field is encountered.

## Using Multiple fields Statements

InData will accept more than one **fields** statement within a prototype, as long as the following restrictions are met:

- ◆ All of the **fields** statements must be outside of conditional prototype statements.
- ◆ No **fields** statement may re-define a field name defined in any previous **fields** statement, placing it in a different position within the field list. However, a **fields** statement may re-define a field name defined in a previous **fields** statement (or built into InData), as long as it is placed in the same position in the field list.

The following illustration provides an example of a prototype which uses multiple **fields** statements:

```
«fields type,lname,fname,addr1,addr2,city,state,zip»
«fields type,corp,div,addr1,addr2,city,state,zip4»
«fields type,edu,school,dept,addr,city,state,zip4»
«if type=1»Dr. «fname» «lname»
«addr1»
«if addr2»«addr2»
«endif»«city», «state» «zip»
«else if type=2»«corp»
«div»
«addr1»
«city», «state»
»
«zip4»
«else if type=3»Dept. Chair., «dept» Dept.
«edu»
«school»
«if addr»«addr»
«endif»«city», «state» «zip4»
«endif — ignore all other record types»
```

The prototype produces mailing labels for three different types of records within the data. It decides which one to produce based upon the value in the **type** field, which is common to all three record structures. The field names defined in the three **fields** statements are each used for a different type of record. Using multiple fields statements in a case like this can make the prototype much more intuitive and easier to read.

## Inserting Styled Text and Text Files

The **put styled** prototype statement can be used to place text files and text containing formatting codes into an InDesign document. This feature allows InData to support XPress Tags styled text in fields, which we mention here for the benefit of former users of QuarkXPress.

This feature uses the **put styled** statement, whose simplest form is:

«**put styled** *expression*»

where *expression* is often (but is not limited to) a field name. For example, the following prototype statement inserts the field **title** into the formatted record and designates it as a **styled** field containing XPress Tags formatting instructions:

«**put styled title**»

You can also designate all of the fields in a prototype as **styled**, with one prototype command:

«**set defaultisstyled to true**»

This statement is often placed near the beginning of the prototype (for example, after the **fields** statement). Note that there is no harm in designating a field without XPress Tags information as **styled** unless it contains angle brackets (< >) or commercial at-signs (@) which might be misinterpreted. You can always use «**put unstyled** *expression*» (discussed later) to insert the given string without style tags interpretation, regardless of the **defaultisstyled** setting.

When the **put styled** statement is used with an expression, the expression may itself contain literal XPress Tags constructs. For example, the prototype fragment

«**put styled** “The book was called <ct:Italic>Ulysses”», and Brian loved it.

would result in the text

**The book was called *Ulysses*, and Brian loved it.**

being placed into the formatted record, even though the prototype writer made an error and forgot the closing <ct:> after the word **Ulysses**. In such cases, the scope of the italics command ends when its prototype statement does, and the literal text following it does not appear in italics.

In general, when InData is importing text containing XPress Tags constructs, the base paragraph and character formats of the text are set to those of the corresponding prototype statement (via either a **put styled** statement or a normal field placeholder if **defaultisstyled** is set to **true**) until an XPress Tags construct changes them. After the tagged text is imported, all formats are restored.

The only limitations on the XPress Tags language are that style sheet definitions **<Define Paragraph Style:....>** and **<Define Character Style....>**, color table definitions **<Color Table:....>** and the hyphenation-related tags are ignored. See the QuarkXPress documentation for a full description of XPress Tags.

## More Complex put styled Statements

The general format of the **put** statement is the following:

«**put** [*how*] *value*»

where *how* may be omitted altogether to simply insert the specified *value* into the formatted record or may consist of one of the keywords **styled**, **unstyled** and **non-styled**, optionally followed by **quoted** or **unquoted**. When keywords are omitted, the **put** statement defaults to **put unstyled unquoted** when **defaultisstyled** is set to **false**, and to **put styled quoted** when **defaultisstyled** is set to **true** (and in both cases the **put** operator itself is optional).

The first set of keywords control whether the imported *value* is interpreted as XPress Tags-styled text (**styled**), as literal text where any XPress Tags constructs are inserted as is (**unstyled**), or as text containing XPress Tags which are to be ignored (**nonstyled**). The **quoted** and **unquoted** keywords control whether quotation mark and double hyphen-to-em dash conversion is performed or not.

Here are some examples (look closely at the double quotation marks):

EXAMPLE	RESULT
«a»	<ct:Italic>This<ct:> ain't--what "I" <ct:Bold>mean<ct:>.
«put styled a»	<i>This ain't—what "I" mean.</i>
«put styled unquoted a»	<i>This ain't--what "I" mean.</i>
«put unstyled a»	<ct:Italic>This<ct:> ain't--what "I" <ct:Bold> mean<ct:>.
«put unstyled quoted a»	<ct:Italic>This<ct:> ain't—what "I" <ct:Bold>mean<ct:>.
«put nonstyled a»	This ain't--what "I" mean.
«put nonstyled quoted a»	This ain't—what "I" mean.

Note that you will often need to insert paragraph-level tags concatenated with the **return** global constant in order to have those settings applied to the imported text. For example, only the prototype form on the right will produce the desired space before paragraph setting in the formatted text:

INCORRECT

```
«put styled "<psb:6>"»
```

CORRECT

```
«put styled "<psb:6>" & return»
```

## Inserting an Entire File of Text

The **put** statement may also be used to insert the contents of a text file into the formatted document, as in this example:

```
«if fileexists("Prelim.TXT")»  
«put styled filecontents("Prelim.TXT")»  
«endif»
```

The **fileexists** function determines whether the specified file exists or not, and the **filecontents** function returns the contents of the specified file. Either one of them can also take an absolute or relative pathname as its argument. Such paths are limited to 255 characters.

Note that **fileexists** is also useful with the **set filename of picture** prototype statement to determine if a given picture file is present before trying to import it (in other words, it obeys the **picturefolders** specification).



# 11

## Hints for Debugging Prototypes

In this chapter, we discuss several strategies for developing prototypes and finding and eliminating errors within them. We present this advice in the form of several suggestions concerning prototype building.

### Test with a Few Sample Records First

Developing and testing complex prototypes is an inherently iterative process. Start your testing by creating or importing a few sample records into a separate text box on the document's pasteboard. If you place the prototype in a text box on the pasteboard as well, then you can easily test it by importing the records from the pasteboard into the main text box on page 1. If you detect a problem, you can quickly modify the prototype and reimport from the pasteboard over the existing records.

It is important that the sample records that you select or generate include all of the features and variations that you expect to see in the actual imported data. Be sure to include sample records that test various boundary conditions: empty fields and/or records, very large fields, and so on.

### Build up the Prototype Gradually

InData wizards can sometimes type in a 50-line prototype and have everything work correctly the very first time. However, the rest of us will be more efficient if we build up the complexity of the prototype gradually.

It is often useful to start by establishing the basic logic of the prototype, especially when this is complicated, using simple literal text or single fields as placeholders for the more complex importing to come, as in the following example:

```
«if very complex condition»it is true«else»it is false«endif»
```

When the condition is working properly, the literal text can be replaced by the desired import fields and expressions.

During the development and debugging process, text color and paragraph space after settings can be used very effectively to isolate problems within complex prototypes. Consider this prototype with its quite complex nested `if` statements:

```
«fields mcode, name, scode, desc, retail, upart, c_num, c_cost, num, cost, m_num,
m_cost, smping, tech_num, tech_cost, status, category, desc2, desc3, unit»
```

```
«if prev name<>name»
«put name marked "A"»
«endif»
```

```
«if pageinfo(1) mod 2 <> 1»
```

```
«if length(desc)<=36» «if c_cost»«c_cost»«else»Call«endif» «upart»
«desc» «if smping»«smping»«else if t_num»«t_num»«else»Call«endif»
«retail»
«else» «if c_cost»«c_cost»«else»Call«endif» «upart»
«desc2» «if smping»«smping»«else if c_num»«c_num»
«else»Call«endif» «retail»
«desc3»
«endif»
```

```
«else»
```

```
«if length(desc)<=36»«upart» «desc» «if smping»«smping»
«else if c_num»«c_num»«else»Call«endif» «retail»
«if c_cost»«c_cost»«else»Call«endif»
«else»«upart» «desc2» «if smping»«smping»«else if
c_num»«c_num»«else»Call«endif» «retail» «if
c_cost»«c_cost»«else»Call«endif»
«desc3»
«endif»
```

```
«endif»
```

We've used underlining and font variations in lieu of different text colors to highlight the structure of this prototype, and have added extra space after the paragraphs containing the outermost `if` statement in the body of the prototype.

This sort of approach has two main advantages:

- ◆ It makes the structure of the prototype clearer, which in turn makes it easier to debug and to modify it.
- ◆ It color-codes the incoming text, so that you can see at a glance which section of the prototype formatted any given piece of data.



## Use Multiple Text Frames for the Prototype

Prototypes like the preceding also benefit from placement within a series of linked pasteboard text frames rather than just a single one. When your prototype contains a lot of variable and property setting statements and initialization and other preliminary statements, placing them within their own text box and ending the final statement in this section with a new box character will make the prototype much more readable.

Sometimes, using even more text frames are useful. For example, here is the way we could format the column wrapping prototype we examined in Chapter 10 (we've truncated a couple of text frames):

We have separated this prototype into a prolog section, an initialization section, loops for each of the columns to be created, and the final section of the outermost repeat loop, each in its own text box. (The illustration cuts off the text frames for the loops for columns 2 and 3 and the final part of the prototype.)

```

«-prolog
«fields firstquote, secondquote, thirdquote
«set itemdelimiter to ""
«set worddelimiters to ""
«set defaultisstyled to true
«-set up column widths
«put 25 into col1width
«put 32 into col2width
«put 15 into col3width

«-initialization
«if recordnumber(1)=true«put 0 into counter«endif
«-copy fields to variables
«put firstquote into col1
«put secondquote into col2
«put thirdquote into col3

«repeat 50 —loop until all columns are exhausted
«    —but use 50 as a fail-safe in case we have a bug >)
«-this is the loop for column 1
«put the number of words in col1 into nwcol1
«repeat —fit as many words from col1 on this line as we can
«    «if nwcol1 <= 1 or length(word 1 to nwcol1 of col1) <= col1width
«        «exit repeat
«    «endif
«put nwcol1 - 1 into nwcol1
«end repeat

«-this is the loop for column 2
«put the number of words in col2 in
«repeat —fit as many words from col2
«    «if nwcol2 <= 1 or length(word 1 to
«        «exit repeat
«    «endif
«put nwcol2 - 1 into nwcol2
«end repeat

«-and this loop is for column 3
«put the number of words in col3 in
«repeat —fit as many words from col3
«    «if nwcol3 <= 1 or length(word 1 to
«        «exit repeat
«    «endif
«put nwcol3 - 1 into nwcol3
«end repeat

«-here is the line that actually outputs text:
«word 1 to nwcol1 of col1» «word 1 to nwcol2 of col2» «word 1 to nwcol3 of col3
«-remove the text we just wrote out from the three variables
«put word nwcol1+1 to (the number of words in col1) of col1 into col1
«put word nwcol2+1 to (the number of words in col2) of col2 into col2
«put word nwcol3+1 to (the number of words in col3) of col3 into col3
«-test whether there is any text left in any of the column variables
«if length(col1) = 0 and length(col2) = 0 and length(col3)=0 —if we're done, output paragraph mark & exit
«exit repeat
«else —we're not done, so just output a line-ending shift-Return
«endif

```

## Make Sure the Data is OK



Sometimes it is the data rather than the prototype which is faulty. The Dump 50 Fields document in the Samples folder can be useful for debugging data files. It will display the contents of up to 50 fields in a data file in an easy to view and understand way. It can help in cases where it is not clear whether a problem is with the prototype or with the data.

The output produced by the prototype is shown below:

```
Record number 1
field 1 is >>Crashaw<<
field 2 is >>Richard<<
field 3 is >>Mr.<<
field 4 is >>Agape Books<<
field 5 is >>928 St. Teresa Terrace<<
field 7 is >>Iconia<<
field 8 is >>NM<<
field 9 is >>72637<<

Record number 2
field 1 is >>Greville<<
field 2 is >>Fulke<<
field 3 is >>Mr.<<
field 4 is >>The Golden Trellis<<
field 5 is >> 876 Caelica Lane<<
field 6 is >>Floor 17<<
field 7 is >>Loredo<<
field 8 is >>TX<<
field 9 is >>56293<<
```

The prototype prints some special characters—>> and <<—before and after the each field in order to make leading and trailing spaces visible.

## Downplaying Prototype Statements



The Spice Catalog document in the Samples folder illustrates a technique which is useful when the prototype is so complex that it threatens to overwhelm the fundamental structure of the formatted result: tabs are overrun, lines wrap, and so on. In these cases, it's hard to know if things are being formatted properly because so much of the prototype text will not appear in the final document (e.g., conditionals). In such cases, making the type size of these extra elements very small—6 or 7 points, or even 3 points for radical cases—can go a long way toward clarifying the structure and look of the prototype:

```

«fields spice,subtype,type1,type2,type3,type4,description»
«if recordnumber(true)=1 or (description is empty and prev description <> empty)»

                                FULL POUND   HALF POUND   2 OUNCES   1 OUNCE

«endif»
«if description is not empty»
«spice», «subtype»           $ «type1»   $ «type2»   $ «type3»   $ «type4»
«description»
«else»
«spice», «subtype»           $ «type1»   $ «type2»   $ «type3»   $ «type4»
«endif»

```

Here, the situation could be more drastic, but we've made the first conditional statement much smaller than the rest of the prototype so that we can see the overall layout more easily.

This prototype also includes an example of a fairly complex conditional in its first **if** statement, where it decides whether or not to include the package size header line or not. It does so only for the very first record (**recordnumber(true)=1**) and when the **description** field for the current record is empty but wasn't empty in the previous record. (The descriptive text is only present in the last variant of each spice, so when we see the above condition, we know we're formatting a new spice.)

Here are some sample formatted records:

	FULL POUND	HALF POUND	2 OUNCES	1 OUNCE
<b>Allspice, whole</b>	\$ 7.90	\$ 3.98	\$ 1.99	\$ .79
<b>Allspice, ground</b>	\$ 8.90	\$ 4.49	\$ 2.29	\$ .89

All of our allspice is from Jamaica, the source for the most flavorful in the world. Allspice is heavily used in Caribbean cooking and in Polish cooking, and it is a popular choice for many baked goods.

	FULL POUND	HALF POUND	2 OUNCES	1 OUNCE
<b>Pepper, Black, whole</b>	\$ 7.90	\$ 3.98	\$ 1.99	\$ .85
<b>Pepper, Black, ground</b>	\$ 6.90	\$ 3.49	\$ 1.79	\$ .69
<b>Pepper, Cayenne, ground</b>	\$ 9.90	\$ 4.98	\$ 2.49	\$ .99
<b>Pepper, Green, whole</b>	\$ 20.50	\$ 10.75	\$ 5.95	\$ 2.25
<b>Pepper, White, whole</b>	\$ 8.90	\$ 4.49	\$ 2.29	\$ .89

Our black, green and white pepper are different varieties of the same plant. The white and black are Tellicherry variety while the green is from Mysore, India. Don't confuse cayenne with the other peppers! Ours is rated at 40,000 Skovel heat units.



# 12

## Automating Document Building

On Macintosh systems, InData supports AppleEvent scripting for automating document building via the AppleScript scripting language. Under Windows, scripting is done with the Windows Automation facility, using VBScript, JScript, Visual Basic or any other supported language.

The first section of this chapter covers the general principles behind InData scripting, and later sections describe the mechanics of controlling InDesign and InData from a script on each platform.

This chapter assumes some very basic familiarity with scripting, on your platform, such as why it's useful and how it works at some very high level.

### Conceptual Overview

The concepts behind InData automation are simple and easily stated: with InData, you interactively name **stories** (and optionally **substories**, which are selections within a named story) within the target InDesign document. Once you've opened a document with named stories, you send InDesign an InData **import data** event to import a given data file, using a specified named prototype story, into a given named target story (and optionally substory) in that document. Note that you must use distinct prototype and target stories when using InData automation; you cannot place your prototype within the target story. You may optionally specify a range of records to import.

InData returns the number of records actually imported in its reply, if the import is successful. Using this return value in your invoking script, you can compute exactly which records were imported, based on the starting record you specified in the original request.

If InData encounters a fatal error (for example, a given prototype named story wasn't found in the current document, or no document is open for an import), it returns a descriptive error string in its reply in such a way that the scripting engine will trap to an error handler with the error description available. You can—and probably always should—use the appropriate language constructs to trap these fatal errors and deal with them yourself (for example, **try** under AppleScript).

For example, the general steps for automated data importing with InData into an InDesign document on the Macintosh using AppleScript are as follows:

- 1M** Create a document containing the desired layout and InData prototype. Assign story names to the text boxes containing the prototype and into which you want to place the formatted records. Define and assign names to substories if appropriate.
- 2M** Open the resulting document in InDesign using the InDesign **open document** AppleEvent.
- 3M** Perform one or more imports into this document using InData's **import data** AppleEvent, making sure that InDesign is the frontmost application.
- 4M** Save the document using InDesign's **save document** AppleEvent.
- 5M** Close the document using InDesign's **close document** AppleEvent.

On a Windows system, the general steps for automated data importing with InData into an InDesign document using Visual Basic are as follows:

- 1W** Create a document containing the desired layout and InData prototype. Assign story names to the text boxes containing the prototype and into which you want to place the formatted records. Define and assign names to substories if appropriate.
- 2W** Open the resulting document in InDesign using the InDesign **Open** method.
- 3W** Perform one or more imports into this document using InData's **ImportFromFile** method, making sure that InDesign is the frontmost application.
- 4W** Save the document using InDesign's **SaveAs** method.
- 5W** Close the document using InDesign's **Close** method.

## Naming Stories

Before you can do any automated document building, you must first build a normal InData template document (either a real InDesign template or a normal InDesign document), and name at least two stories in the document: a prototype story and a target story for data importing. Note that InData makes no distinction between prototype and target stories in the naming process—both types are just ordinary named stories.

You can name as many stories as you like, if you need multiple sets of prototype and target stories. You will use substories if you need to import into different portions of the same target story (discussed in the next subsection).

You name stories with the **InData=>Name Story...** menu item, which is enabled only when you have a text frame selected with the content tool, and when you're viewing ordinary document pages, not master pages. The **Name Story...** menu item will be checked if the current story already has a name, and you can change or remove the name, again using the **Name Story...** dialog:



The **Name of story** field is used to assign or change the name of the current story, and the **Delete Name** button may be used to remove the name from the current story.

Story names are *case-sensitive* and must be unique within a given document.

## Naming Substories

InData supports the naming of **substories**: named text selections within a story. This facility allows you to aim an InData import at a specific subsection of a named target story. This is a useful feature when you are building a complex document out of many pieces, some of which reside in the same text flow as others. Named substories retain their identity (and name) even after the import is finished, so you can rename them or even re-import over an already-imported substory. (However, as is true in general, InData won't delete any extra pages if the newly-imported data is shorter than the previously-imported data).

You name substories by selecting some text within a named story, and then selecting the **InData=>Name Substory...** menu item. This resulting dialog is almost identical to the **Name Story...** dialog, and it functions the same way for renaming and deletion. Substory names must be unique within a story, and are also case-sensitive.

Named substories use InDesign hidden text enclosing the selected text, so you must be careful to avoid accidentally deleting the substory names when editing your template (unless you're deleting the whole substory intentionally).

## Finding Stories and Substories by Name

The **InData=>Find Story/Substory...** menu item (which is enabled only when you have a document open and are in content tool mode) allows you to search for a given story (and optionally substory) by name. If there is a story selected and it has a name, it is initially placed in the **Find Story** field, to make it easy to find substories without remembering the exact name of the current story.

If you don't want to find a substory, leave the **and Substory** name field empty.

## Building Complex Multi-Part Documents

When building complex multi-part documents with InData automation, you'll need to set up your template document carefully. In particular, if you need to perform multiple imports of unknown length into multiple target stories in a single document, make sure that when you create each target story, you do so by dragging a new master page with an automatic text chain on it (the first frame of which you will eventually name as your target story).

## Using InData with AppleScript (Macintosh)

The InData **import data** function, when inspected with the AppleScript Editor's **Open Dictionary...** command (select the InData.pln plug-in file when prompted), is quite simple:

**import data from file:** Import (and format) a raw data file into a target story (and optionally sub-story) using a prototype story.

```
import data from file alias
into story string
[substory string]
using prototype story string
[starting with record integer]
[ending with record integer]
```

Result: integer

Thus, InData adds one new scripting command to InDesign: **import data from file**. Given a file, a target story name, an optional target substory name, a prototype story name, and optional starting and ending record numbers, InData does the import and returns the number of records imported.

The given file *alias* may also be a full pathname string (because of the built-in AppleScript conversion from pathname to alias), while each of the target story, optional substory, and prototype story names is just a string containing the name of the story or substory in question. Note that an empty substory name means to insert text at the end of the target story (regardless of whether there are any defined substories within the specified target story).

For InData to work correctly, you should always give an **activate** command before doing any **import data from file** commands. In other words, your script should always be of the form:

```
tell application "InDesign 2.0"           Use the appropriate version number!
    activate
    import data from file ...
    import data from file ...
end tell
```

Otherwise, if InData finishes an import while InDesign is in the background (is not the frontmost application), it will hang until you manually bring InDesign to the front.



Note that you can use the AppleScript statement:

**tell me to activate**

at any point to bring the script editor (or the script itself, if it's running as an application or droplet) back to the foreground.

## An Example Script



The AppleScript compiled script file BuildDemo in the AppleScript subfolder of the Scripting Examples folder illustrates various ways you can import data with InData from AppleScript scripts. To see it work, double-click the script file and then press the **Run** button in the resulting Script Editor window. This script illustrates importing records from several sources using multiple raw data files, prototype stories and target stories and substories.

## Importing Large Data Files in Batches



The script file MailMergeDemo in the Scripting Examples folder illustrates one method for processing very large data sets in batches that InDesign can handle. Here are the key parts of this script to which we've added additional annotations:

```
--AppleScript MailMergeDemo, by Shane Stanley
set complexityFactor to 10
-- complexityFactor is a rough figure related to how long it takes to print each page,
-- and is used to set the timeout for printing. Adjust it (upwards) if you get
-- timeout errors when printing.
set templateDoc to (choose file with prompt "Loacte the InData template." of type {"InDd"})
set dataDoc to (choose file with prompt "Where is the data file?" of type {"TEXT"})
-- prompt for batch size and saving preferences
set batchInfo to (display dialog ~
    "How many in a batch? Do you want to save each batch ...?" buttons ~
    {"Cancel", "Print", "Save"} default button "Save" default answer "50")
try
    -- save batch size, catching error for bad input
    set batchSize to (text returned of batchInfo) as integer
on error
    display dialog "Illegal entry: ... must be a whole number." buttons {"OK"} ~
        default button "OK"
    error number -128
end try

-- save batch size and do other initialization
copy batchSize to recsImp
set batchDo to button returned of batchInfo
set firstRec to 1

tell application "InDesign 2.0"
    activate
    -- loop over batches as long as batches are successfully imported
    -- loop will end on any error or when an imported batch is shorter than
    -- the specified batch size (hopefully, the last batch)
```

```

repeat while recsImp = batchSize
    -- set the lastRec variable to the final record in this batch
    set lastRec to firstRec + batchSize - 1
    open templateDoc
    import data from file (dataDoc as text) ¬
        using prototype story "proto 1" into story "story 1" starting ¬
        with record firstRec ending with record lastRec
    --save the value returned by the import data operation
    set recsImp to result
    if (recsImp as integer) > 0 then
        --import succeeded; reset lastRec just in case to reflect
        --actual number of imported records
        set lastRec to firstRec + (recsImp as integer) - 1
        --save or print document if user requested it
        if batchDo = "Save" then
            save document 1 in file ((templateDoc as text) & ¬
                " (" & (firstRec as text) & "-" & (lastRec as text) & ".")
        else
            with timeout of (120 + ((recsImp as integer) * ¬
                (complexityFactor))) seconds
                print document 1
            end timeout
        end if
        close document 1 saving no
        --reset firstRec variable to the next record to be imported
        set firstRec to lastRec + 1
    else
        -- import failed so close document
        close document 1 saving no
    end if
end repeat
end tell

```

## Using InData with Windows Automation (Windows)

The automation process under Windows is conceptually similar to that just described on the Macintosh. The key call is to the method (function) **InDataImportFromFile**, which has the following general syntax:

```

long InDataImportFromFile (
    [in] VARIANT fileName,           File containing data to import.
    [in] BSTR prototype StoryName,   Story holding the prototype.
    [in] BSTR targetStoryName,       Place imported records in this story.
    [optional, in] BSTR targetSubStoryName, ... and substory (if specified).
    [optional, in] long startingRecordNumber, Specify optional starting and/or
    [optional, in] long endingRecordNumber; ... ending record numbers.

```

Here is a simple Visual Basic code snippet which calls this method:

```

Private Sub Command1_Click()
    Set myInDesign = GetObject ("", "InDesign.Application")
    myInDesign.Activate
    myInDesign.Open ("c:\doc\mydoc.indd")

```

```
On Error GoTo Myerrorhandler
```

```
' count is the number of records processed
```

```
count = myInDesign.InDataImportFromFile ("c:\doc\mydata.txt","proto_box","target_box")
```

```
Exit Sub
```

```
Myerrorhandler:
```

```
MsgBox ("InData: " & Err.Description)
```

```
End Sub
```

The final portion of this code excerpt defines a simple error handler.

## An Example Script



The Visual Basic project file, BuildDemo.vbp, in the Visual Basic subfolder of the Scripting Examples folder, illustrates various ways you can import data with InData from Visual Basic scripts. To see it work, double-click the project file to open it, and then run it by selecting **Run=>Start** from the menu (or pressing F5).

Once you've started running the project, the following simple form will appear:



Click the button to start the demonstration. This script illustrates importing records from several sources using multiple raw data files, prototype stories and target stories and substories.

Note that the code is attached to the button on the form, and the form itself is stored in the file BuildDemo.frm. Within the Visual Basic editor, you can view the code by selecting **View=>Code** from the menu if it is not initially visible.



# 13

## InData Reference

This chapter serves as a reference to the InData menu selections, dialogs, and prototype language. It opens with a table of keyboard equivalents for various special characters useful to InData and InDesign users.

### Entering Special Characters

CHAR.	PURPOSE	KEYSTROKE
¶	End current line and start new para.	Return
↵	New line: force a line break, but don't start new para. (aka "hard return").	Shift-Return
⇥	New column: force a column break.	Enter (on numeric keypad)
⇧⇥	New (or next) frame: force text into the next text frame in this text flow.	Shift-Enter (on numeric keypad)
A	Page number: insert number of current page here.	Opt-Command-N (Mac) Ctrl-Alt-N (Win)
«	Left chevron (guillemet): begin InData prototype element.	Option-\ (Mac) Alt-0171 (Win)
»	Right chevron (guillemet): end InData prototype element.	Shift-Option-\ (Mac) Alt-0187 (Win)
»	Tab: move to the next tab stop (displays as a blue right chevron).	Tab key
!=	Not equals sign.	Option-= (Mac) Alt-= (Win)
>=, <=	Greater/less than or equals signs.	Option->, < (Mac) Alt->, < (Win)

## The InData Menu

This section documents each of the items on the InData menu.

**Import from File...**

Import data from an external data file. A file selection dialog will appear, allowing you to specify the location of the data file.

**Import from Clipboard...**

Import data records currently on the system clipboard. This option is disabled if there is no text on the clipboard.

**Import from Pasteboard...**

Import data records from a unique text story (composed of one or more text frames) completely on the pasteboard of the current spread. (A text story previously designated as the prototype story may also be present on the pasteboard when this option is chosen, without confusion.)

All of the **Import...** selections are disabled unless a target text flow has been designated with an insertion point or text selection (which implies the content tool is selected).

**Make Header/Footer**

Designate the currently selected text—which should be in a master page text frame that is not part of the main document text flow—as a mark reference. The resulting dialog will allow you to specify the corresponding mark name and other characteristics of this reference.

**Update Headers/Footers...**

Update previously generated headers and footers to reflect minor editing changes made to the imported text. The insertion point must be in the text flow containing the already-imported data.

**Use Story as Prototype**

Designate the currently selected text frame as the prototype. Any subsequent designation of a different story (including the main text flow) will supercede the currently designated prototype.

**Name Story...** Assign a name to a text flow within the document.

**Name Substory...**

Assign a name to a selection within a named story.

**Find Story/Substory ...**

Locate a specified story and possibly substory within the current InDesign document.

<b>Preferences</b>	View or alter InData preferences (its submenu is described below). If selected while a document is open, preferences apply to that document alone. If selected while no document is open, global preferences are shown or set.
<b>About...</b>	Display InData version, serial number, personalization data, and product credits.

### InData Preferences Submenu

<b>Data...</b>	Open the <b>Data Preferences</b> dialog, which allows you to specify preferences for the format of the data records.
<b>Range...</b>	Open the <b>Range Preferences</b> dialog, where you can specify preferences for the starting and ending records to import.
<b>View...</b>	Open the <b>View Preferences</b> dialog, in which you can specify preferences for how often (or whether) InData should update the screen while importing records.
<b>General...</b>	Open the <b>General Preferences</b> dialog, where you can specify preferences for some InData picture handling and importing auto-start behavior (among other things).

## InData Control Panel Buttons

<b>Data...</b>	Open the <b>Data Preferences</b> dialog, where you can specify the characteristics of the data records. (Enabled only before importing begins.)
<b>Range...</b>	Open the <b>Range Preferences</b> dialog, where you can specify the starting and ending records to import. (Enabled only before importing begins.)
<b>View...</b>	Open the <b>View Preferences</b> dialog, where you can specify how often InData updates the document window during data importing. (Enabled only before importing begins.)
<b>Start</b>	Begin data importing. This button changes to <b>Pause</b> once importing begins, and it may then be used to temporarily halt the import process.
<b>Cancel</b>	Dismiss InData before starting the import operation. After starting, this button changes to <b>Stop</b> , which halts data importing immediately.
<b>Show</b>	Immediately update the document view. (Disabled if you've chosen the <b>Show document changes every __ records</b> view option.)

## Data Preferences Panel

### Basic format Pop-up Menu

#### Comma-delimited

Fields are separated by commas; all other settings as for tab-delimited below.

**Tab-delimited** InData recommended format: Fields are separated by tabs, subfields (from repeating/multivalued fields) are separated by the ASCII “group separator” character (29), and records are separated by carriage returns. Quoted fields start and end with double quotation marks; quotation marks within quoted fields are doubled.

#### Microsoft Word™ merge comma-delimited

Same as comma-delimited except that the first record is assumed to be a header record and is automatically skipped during import.

#### Microsoft Works™ tab-delimited

Same as tab-delimited except that the first record is assumed to be a header record and is automatically skipped during import.

#### Custom

Any format differing from the four standard formats described above.

### Other Format Definition Fields

All field values may be set either to a literal ASCII character, to a 2- or 3-digit ASCII code number, or to a blank (empty); the latter means that the corresponding function is disabled.

**Field delimiter** Character separating data fields (defaults to comma or tab).

#### Subfield delimiter

Character separating subfields in repeating records (usually data from FileMaker). It defaults to the group separator: ASCII 29.

#### Record delimiter

Character separating records in the data file (defaults to carriage return).

#### Quote character

Character surrounding fields containing embedded field, subfield, or record delimiters (defaults to straight double quotation mark).

#### First record is header

If checked, ignore the first record in the data file automatically



(independently of the **Range Preferences**). The default is unchecked.

**FoxBase/FoxPro-style quoting (not doubled)**

Prevents adjacent double quotes ("" ) within fields from being interpreted as a single double quote (").

**Other Fields**

These field values may be set to **Return**, meaning replace the character by a paragraph mark (carriage return), **Shift-Return**, meaning replace the character by the new line character which forces a line break (Shift-Return), **Space** (the space character), or **Nothing**, meaning discard the character while importing.

**Carriage return in data becomes**

Translate ASCII carriage returns to this character; the default is the paragraph mark (**Return**).

**Vertical tab in data becomes**

Translate vertical tabs (ASCII 11) to this character; the default is the new line character (**Shift-Return**).

**Character set**

Specify the character set for imported data: **Windows**, **Macintosh** or **Unicode**. InData autodetects the UTF-8 and UCS-2BE/LE encodings when **Unicode** is selected.

## View Preferences Panel

**Hide document window**

Show absolutely no window updates during the entire import operation. This is the fastest import mode, by far.

**Don't show document changes**

Leave the document window visible but mostly unchanging during importing, and update it only after importing all records. (Changes may show up anyway, when importing pictures, due to InDesign machinations.)

**Show document changes every \_\_ records**

Update the document window after importing each specified number of records (the default of 10 is pre-entered into the field).

## Find Story/Substory Dialog

The two fields in this dialog allow you to specify the story and optionally substory for which you want to search. When the insertion point or selection is within a named story, then that field will set accordingly when this dialog opens.

## General Preferences Panel

### Default picture position

Where and how imported pictures should be placed within their picture frames. Its pop-up menu options are:

**Top left:** Place the picture's upper left corner in the upper left corner of the picture frame.

**Center:** Center the picture within the picture frame.

**Center, Size to Fit:** Scale the picture to fit into the picture frame exactly.

**Center, Size to Fit, w/o Distortion:** Scale the picture to fit the picture frame, maintaining its aspect ratio (original proportions), and then center it within the picture frame.

**Size Frame to Picture:** Shrink the anchored picture frame to fit its contents (once they are imported), obeying any scaling and margin picture frame properties and ignoring any offset picture frame properties.

**Size to Fit Horizontally, then Size Frame Vertically to Picture:** Size the picture itself in its anchored picture frame to fit horizontally (i.e., to fill the frame in the "x" direction) once the picture is imported, sets the picture's y scale to match its x scale (as determined above by making it fit horizontally), and then shrink the anchored picture frame itself vertically to make it fit the contained picture.

### Start automatically after source selection

If checked, start importing immediately as soon as the location of the data is known (i.e., don't wait until the **Start** button is pushed).

### Keep importing after missing pictures

If checked, continue importing records despite missing picture files (by default, InData halts when it can't find a requested picture file).

### Don't automatically update headers/footers

Controls whether headers and footers are updated automatically at the conclusion of an import operation. By default it is unchecked.

**I'm an expert user (suppress warnings)**

If checked, then various warning messages that occur when the imported data overflows its text chain are not displayed. It is unchecked by default.

**Maximum length of variables**

Controls the maximum number of characters that you can place into a variable with InData's **put** prototype statement (in multiples of 1024 bytes). You should not normally need a value outside of the range of 2 to 8 for this setting.

## Make Header/Footer Dialog

**Make reference to first/last**

Whether the first or last occurrence of the marked text on the page or spread should be placed in the header/footer.

**text marked \_\_\_\_\_**

Mark name of the desired field or expression as defined in a **put ... marked "X"** statement in the prototype (the same *x* should be filled in here).

**on the current page/spread**

Whether the first/last setting refers to the current page or to the current spread.

**Limit initial search for the first occurrence of marked text**

Specifies a range of lines in which the first marked text must be found in order for it to be used in the header or footer. By default, marked text found anywhere on the page/spread will be used.

**Reference text marks from previous pages**

If checked, use the most recent marked text from previous pages, if none falls on this page/spread, or if none is found a limited initial search.

**appending \_\_\_\_\_**

If marked text from a previous spread is used on this page, append the indicated string to it. Formatting with any InDesign Tags character style tags is supported within this string.

## Name Story and Name Substory Dialogs

These dialogs both have a single field into which the desired name may be typed. Their **Delete Name** button may be used to remove the current name from any current story or substory.

## Range Preferences Panel

<b>Import record</b> _____	First record to import. Numbers begin at 1.
<b>through</b> _____	Last record to import. Can be either a record number, or one of the keywords <b>end</b> , <b>all</b> , <b>last</b> , <b>final</b> or <b>•</b> , all of which mean to import up through the end of the data.

## InData Prototype Elements

In the descriptions that follow, the following symbols and typographic conventions are used in syntax descriptions:

<b>boldface</b>	InData command or keyword to be typed exactly as shown.
<i>italic</i>	A parameter: something you must fill in when you use the command or operation. For example, <i>field</i> means you must enter the name of a field.
[ ]	Portions of statements or expressions in italicized brackets are optional.
<b>a   b</b>	Select one of the choices separated by the vertical bar.

In general, all statement arguments may be constants or expressions. For example, the first two arguments used by **character**—the starting and ending character indices—may be:

numbers:

**char 1 to 3 of a**

an expression which evaluates to a number:

**char count to count+1 of a**

a function which returns a number:

**char 1 to length(a) of a**

or combinations of these:

**char count+1 to length(a)-count of a**

Similarly, the third argument to **character**—the expression to extract characters from—may be:

a literal string:

**char 1 to 3 of "hello"**

a field name:

**char 10 to length(corp) of corp**

or an arbitrarily complex expression:

**char 1 to 3 of word 1 to 2 of subfield 8 of previous phone**

## Prototype Statements

### ask

SYNTAX: «ask *prompt-expression* [ with *answer-expression* ] »

USE: User querying during importing. The prompt will appear in a dialog containing an input field and the buttons **OK**, **Cancel**, and **Stop**. The answer is used to pre-fill in the input field (i.e., used as a default). The information entered by the user is placed into the variable **it**, which may be used later within the prototype. If the user pressed **Cancel**, **it** will be empty. If the user presses **Stop**, the whole InData import will be stopped immediately.

EXAMPLE: «ask "Enter first name:" » «it»  
«ask "Enter job title:" with "Engineer" » «it»

### exit

SYNTAX: «exit»

USE: Immediately stops data importing.

EXAMPLE: «if lname > "M" » «exit» «endif»

### exit repeat

SYNTAX: «exit repeat»

USE: Immediately exit from the innermost repeat loop.

EXAMPLE: «repeat iv=1 to length(a) » «if char iv of a = " \_ " »  
«exit repeat» «]» *additional statements* «end repeat»

### fields

SYNTAX: «fields *name1, name2, ..., namen*»

ALT. FORMS: «columns *name1, name2, ..., namen*»

USE: Used to name the fields in the data records. Fields can be skipped using 2 or more consecutive commas. Field names may include letters, number, and the underbar character (\_); they must begin with a letter or underbar. The automatic field names **a** to **z** may not be redefined.

EXAMPLE: «fields last,first,title,phone»  
«columns price, cat\_num, descr»

### if

SYNTAX: «if *condition* » *statements* [ «else» *statements* ] «endif»

ALT. FORMS: «[ *condition* » ... «]» *if ... endif*

«[ *condition* » ... «|» ... «]» *if ... else ... endif*

«if *condition1* » ... «else if *condition2* » ... «else» ... «]» *case*

USE: Conditionally includes data within imported records. If *condition* is true for a record, then the *statements* will be carried out for that record. Otherwise, any statements in the **else** clause will be carried out. Conditions must evaluate to a boolean (**true** or **false**) value. There can be any number of **else if** clauses in a chain.

EXAMPLE: «if lname » «lname » «else » «corp » «endif»

**next**

SYNTAX:

**«next»**

USE:

Immediately returns to the beginning of the prototype as if the end of the prototype had been reached, and starts processing the next record. Contrast this with the **read** statement, which reads the next record but continues processing the prototype.

EXAMPLE:

**«if balance < 0» «next» «endif»****next repeat**

SYNTAX:

**«next repeat»**

USE:

Immediately begins the next **repeat** loop iteration as if the end of the **repeat** loop had been reached.

EXAMPLE:

**«repeat 5» «if lname Ž "M"» «next repeat» «endif»  
additional statements«end repeat»**

**open**

SYNTAX:

**«open filename»**

USE:

Open a data file from within the prototype, suppressing the usual input file selection dialog. The given filename can be either relative to the current document's folder, or it can specify an absolute location for the data file.

EXAMPLE:

**«open "Poole employees"»  
«open "HD80:June Catalog:Parts"»**

**put**

SYNTAX:

**«put expression»**

ABBREVIATION:

**«expression»**

USE:

Inserts the expression (often a field) as text into the formatted records.

EXAMPLE:

**«put lname» or «lname»  
«put "(415) "» or «"(415) "» or (415)**

**put into**

SYNTAX:

**«put expression into variable»**

USE:

Sets the value of the given *variable* to the results of *expression*. Before a variable is set for the first time, it is empty.

EXAMPLE:

**«if counter is empty» «put 1 into counter»  
«else» «put counter+1 into counter» «endif»**

**put marked**

SYNTAX:

**«put expression [ hidden] marked "mark-name"»**

USE:

Inserts the *expression* as text into the formatted records, and marks it for potential use in a running header or footer. The *mark-name* must be a single, case-insensitive character. If **hidden** is included, then the resulting text will not be visible in the for-

matted records, so the **put ... hidden** should normally be placed on the same line as a related normal field placeholder.

EXAMPLE:

```
«put lastname marked "A"»
«put hidden char 1 of ln marked "L"»«ln»
«put char 1 of ln marked "L"»«put char 2 to length(ln) of ln»
```

*Examples 2 and 3 have an identical effect.*

### put styled

SYNTAX:

USE:

```
«put [ styled | unstyled | nonstyled ] [ [un]quoted ] expression»
```

When the keyword is **styled**, this statement inserts the result of *expression*, processed as XPress Tags styled input, into the formatted records (*expression* is often a field name). The first set of keywords controls whether the imported *expression* is interpreted as XPress Tags-styled text (**styled**), as literal text where any XPress Tags constructs are inserted as is (**unstyled**), or as text containing XPress Tags which are to be scanned but otherwise ignored (**nonstyled**). The **quoted** and **unquoted** keywords control whether quotation mark and double hyphen-to-em dash conversion is performed or not.

EXAMPLE:

```
«put styled unquoted book_title»
«put styled "<ct:Bold>" & a & "<ct:>"»
```

### read

SYNTAX:

USE:

```
«read»
```

Immediately reads the next data record and then continues processing the prototype after the **read** statement. Contrast this with the **next** statement, which skips the rest of prototype processing for the current record.

EXAMPLE:

```
«if continue_flag»«read»«endif»
```

### repeat

SYNTAX:

USE:

```
«repeat count»...«end repeat»
«repeat with var = start [down] to end»...«end repeat»
«repeat while | until condition»...«end repeat»
«repeat forever»...«end repeat»
«repeat»...«end repeat»
```

Creates a loop within an InData prototype. When a *count* is given as **repeat**'s argument, the loop runs that many iterations unless an **exit repeat** statement is encountered. **repeat while** continues as long as *condition* remains true, and **repeat until** continues until *condition* becomes true. **repeat with** is used to create an ascending or descending (by including **down**) indexed loop. **repeat forever** repeats continuously and can only be exited by **exit repeat** or **next repeat**. **repeat** is equivalent to **repeat forever**.

EXAMPLE:

```
«repeat 1000» loop body«end repeat»
«repeat while dept≠next dept» loop body«end repeat»
«repeat with ind=10 down to 1» loop body«end repeat»
```

**set defaultisstyled**SYNTAX: «**set defaultisstyled to** *true-or-false*»

USE: Allows you to set whether or not all field placeholders support XPress Tags formatting on a prototype-wide basis. In other words, if this variable is set to true, then any field placeholder becomes equivalent to its **put styled** version: «**a**» is equivalent to «**put styled a**», for example. The default setting is **false**.

EXAMPLE: «**set defaultisstyled to true**¶**set filename of picture**ABBREVIATION: **set fn of pic**SYNTAX: «**set [the] filename of picture** *number to path*»

USE: Specifies the picture file location as *path* for the picture frame indexed by *number* (starting with 1, counting from the start of the prototype, ignoring the effect of conditionals). The path expression is often a fieldname, meaning that the picture filename for each record is taken from that field.

EXAMPLE: «**set fn of picture 2 to pic2name**¶**set itemdelimiter**SYNTAX: «**set itemdelimiter to** *char*»

USE: Specifies the character that separates items within an expression. The default character is a comma.

EXAMPLE: «**set itemdelimiter to “/”**»**set master**SYNTAX: «**set [the] [first]master of this page | spread to** *mastername*»

USE: Assigns a master page/spread to the current page or spread during importing. When multiple **set master** statements are encountered on the same page/spread, the final one takes precedence. When multiple **set firstmaster** statements appear on a page/spread, the first one takes precedence.

EXAMPLE: «**set master of this page to “Master A”**»**set picturefolders**SYNTAX: «**set picturefolders to** “*list-of-folders*”»

USE: Specifies locations for InData to look for picture files in addition to the location specified in the picture filename (which defaults to the current folder). The *list-of-folders* should be separated by commas. Folder names may be absolute (Data:Pictures: means the folder Pictures on disk Data) or relative (Old:Pictures: means the folder Pictures in the folder Old in the same folder as the current document). Under Windows, use DOS-style pathnames (e.g., C:\Data\Pictures\ and My\_Pix\). The terminal colon/slash is *required*.

EXAMPLE: «**set picturefolders to “:Pictures:,Disk 2:Pix:”**¶  
 «**set picturefolders to “C:\Pictures\,D:\Pix\,My\_Pix\”**¶



**set pictureimportcrop**

AVAILABILITY: *InData 2.0 only.*

SYNTAX: «**set pictureimportcrop** [of picture *number*] to *n*»

USE: Specifies the cropping for an imported PDF file page. The value *n* runs from 0 to 3. **0** uses the InDesign global default cropping (as last set in the **PDF Import Options** dialog); **1** crops to the media; **2** crops to the page; and **3** crops to the content (bounding box). Any other value is ignored.

EXAMPLE: «**set pictureimportcrop of picture 2 to 2**»

**set pictureimportpage**

AVAILABILITY: *InData 2.0 only.*

SYNTAX: «**set pictureimportpage** [of picture *number*] to *n*»

USE: Specifies the page of a PDF file to import as a picture. The value *n* should be a non-negative integer; invalid values default to 1. A value of 0 has the special meaning of the InDesign global default: the page last set in the **PDF Import Options** dialog.

EXAMPLE: «**set pictureimportpage of picture 2 to 10**»

**set pictureposition**

ABBREVIATION: **set picpos** [of pic]

SYNTAX: «**set [the] pictureposition** [ of picture *number*] to *keyword*»

USE: Specifies how a given picture (or all pictures if the of picture clause is omitted) should be placed within their picture frames. The *keyword* must be one of: **toleft**, **center**, **fit**, **aspectratiofit**, **framefit**, and **fithframev** which have the same meanings as the corresponding items on the **Default picture position** menu in the **General Preferences** dialog.

EXAMPLE: «**set picpos of picture 1 to center**»

**set worddelimiters**

SYNTAX: «**set worddelimiters to** *chars*»

USE: Specifies characters that separate words within expressions. The defaults are space and all characters in the ASCII character set below it (line feed, return, tab, etc.).

EXAMPLE: «**set worddelimiters to “.,/”**»

**set wordcharacters**

SYNTAX: «**set wordcharacters to** *chars*»

USE: Specifies characters that are considered part of words. All other characters are considered word delimiters.

EXAMPLE: «**set wordcharacters to “abcdef”**»

The **set** statement also has a number of other options which allow you to specify various InData dialog settings from within the prototype. They are summarized in the following table.

KEYWORD	ARGUMENT	EXAMPLE
<b>autostart</b>	<i>true/false</i>	«set autostart to true»
<b>clipboard</b>	<i>true/false</i>	«set clipboard to false»
<b>endrecord</b>	<i>number</i>	«set endrecord to 1000»
<b>fieldquote</b>	<i>character</i>	«set fieldquote to ""» «set fieldquote to "\"" <i>disabled</i>
<b>fieldseparator</b>	<i>character</i>	«set fieldseparator to ","»
<b>fldsep</b>		«set fldsep to ","»
<b>filetype</b>	<i>type</i>	«set filetype to "Tab-delimited"»
KEYWORD	ARGUMENT	EXAMPLE
<b>ignoremissingpictures</b>	<i>true/false</i>	«set ignoremissingpictures to true»
<b>recordseparator</b>	<i>character</i>	«set recordseparator to "\n"»
<b>recsep</b>		«set recsep to "\n"»
<b>startrecord</b>	<i>number</i>	«set startrecord to 100»
<b>subfieldseparator</b>	<i>character</i>	«set subfieldseparator to ","»
<b>subfldsep</b>		«set subfldsep to ","»
<b>viewfrequency</b>	<i>number</i>	«set viewfrequency to 50» «set viewfrequency to 0»

## Expression Operators

The following InData operators may be used in constructing expressions. Each one is described individually below, including its required and optional arguments.

### character

ABBREVIATION:

**char**

SYNTAX:

**character** *start* [ *to end* ] **of** *expression*

USE:

Used to extract a range of characters from a string. If the **to** clause is omitted, then one character is extracted. Character indices start at 1; any index less than 0 is treated as if it were 1, and any index greater than the length of *expression* is treated as if it were the length of *expression*. The expressions *start* and *end* must be integer (whole number) character index expressions. Use the **length** function for *end* to extract through the end of a string. Expressions involving **char** evaluate to a character string.

EXAMPLE:

**char 1 to 3 of phone**

**char length(price)-2 to length(price) of price**

*Extracts up to 3 characters from the end of the price field.*

**chartonum**

SYNTAX: **chartonum**(*char*)  
USE: Returns the ASCII character number of the character given as its argument as a string.  
EXAMPLE: **chartonum("A")**

**dec2frac**

SYNTAX: **dec2frac**(*string*)  
USE: Converts the specified character string to a fraction, returning the whole number and fractional parts as separate words.  
EXAMPLE: **dec2frac("1.375")**

**downcase**

SYNTAX: **downcase**(*string*)  
USE: Converts a character string expression to lowercase.  
EXAMPLE: **downcase(first & last)**

**filecontents**

SYNTAX: **filecontents**(*path*)  
USE: Returns the contents of the specified text file as a text string. The path may be absolute or relative to the current directory and should be in the proper format for the computing environment (Macintosh or Windows).  
EXAMPLE: **filecontents("new.txt")**

**fileexists**

SYNTAX: **fileexists**(*path*)  
USE: Returns true or false depending on whether the specified file exists or not (may be imported as a picture or with **filecontents** if text). The path may be absolute or relative to the current directory and should be in the proper format for the computing environment (Macintosh or Windows), and is interpreted relative to the current picture paths.  
EXAMPLE: **fileexists("new.txt")**

**fileinfo**

SYNTAX: **fileinfo**(*j,n*)  
USE: Returns information about the current document file (when *j* is 0) or data file (when *j* is 1). The information returned depends on the value of *n*, 1 returns the base file name; 2 returns the full pathname.  
EXAMPLE: **«put fileinfo(1) into fname»**

**frameinfo**

SYNTAX:

**frameinfo(*n*)**

USE:

Returns information about the text frame at the current insertion point. The information returned depends on the value of *n*, **1** returns an ephemeral unique identifier for the current text frame (only unique for the given document session); **2** returns the 1-based index of the current frame in the text thread on the current page; **3** returns the 1-based index of the current frame in the text thread on the current spread.

EXAMPLE:

**«put frameinfo(1) into frameid»**

**item**

SYNTAX:

**item *start* [*to end*] of *expression***

USE:

Used to extract a range of items from a string, where items are delimited by commas or the character specified via **set itemdelim**. If the **to** clause is omitted, then one item is extracted. Item indices start at 1; any index less than 0 is treated as if it were 1, and any index greater than the number of items in *expression* is treated as if it were that value. The expressions *start* and *end* must resolve to integers (whole numbers).

EXAMPLE:

**item 1 to 3 of contract**

**length**

ALT. FORMS:

**the length of**

SYNTAX:

**length(*expression*)****the length of *expression***

USE:

Returns the length in characters of the specified *expression* (returns 0 if *expression* is empty/null). The **length** function returns an integer value which is 0.

EXAMPLE:

**«if length(phone) < 7»(415) «endif» «phone»**

*Adds an area code to phone numbers shorter than 8 characters.*

**line**

SYNTAX:

**line *start* [*to end*] of *expression***

USE:

Used to extract a range of lines—strings separated by carriage returns—from a string. If the **to** clause is omitted, then one line is extracted. Line indices start at 1; any index less than 0 is treated as if it were 1, and any index greater than the number of lines in *expression* is treated as if it were that value. The expressions *start* and *end* must be integer (whole number) expressions.

EXAMPLE:

**line 3 to 6 of comments**

**next**

SYNTAX: **next** *fieldname*  
 USE: Refers to the contents of the field named *fieldname* in the next (upcoming) record.  
 EXAMPLE: «if dept  $\neq$  next dept» «dept» «endif»

**number**

SYNTAX: **/the/number of** *chunks in expression*  
 USE: Returns the number of entities of the specified type in *expression*. *Chunks* must be one of **chars**, **words**, **lines** or **items**.  
 EXAMPLE: **the number of words in comment**

**numtochar**

SYNTAX: **numtochar(integer)**  
 USE: Converts an integer in string form to the corr. ASCII character.  
 EXAMPLE: **numtochar(68)**

**offset**

SYNTAX: **offset(pattern, expression)**  
 USE: Searches for the *pattern* within *expression*. If it finds it, **offset** returns the character number where it starts, or 0 if it's not found. The **offset** function returns an integer value in all cases.  
 EXAMPLE: «if offset(",phone) = 0» («char 1 to 3 of phone») «char 4 to length(phone) of phone» «else» «phone» «endif»  
*Adds parentheses around the area code in a phone number if not present.*

**pageinfo**

SYNTAX: **pageinfo(n)**  
 USE: Returns information about the current page at the current insertion point. Which information is returned depends on the value of *n*, which ranges from 1 to 5: **1** returns the absolute page number in document; **2** returns the absolute spread number in document; **3** returns the relative page number within the current section of the document; **4** returns the page location within the current spread (where the left page=1, and the right page=2); **5** returns the name of the currently-applied master spread.  
 EXAMPLE: «put pageinfo(2) into spreadnum»  
 «if pageinfo(5) = "Master A"» ... «endif»

**previous**

ABBREVIATION: **prev**  
 SYNTAX: **prev** *fieldname*  
 USE: Refers to the contents of the field named *fieldname* in the previous record.  
 EXAMPLE: «if dept  $\neq$  prev dept» «dept» «endif»

**recordnumber**

SYNTAX:

**recordnumber**(*true-or-false*)

USE:

Returns the record number of the current record. If its argument is the keyword **false**, then it returns the absolute record number in the data file, regardless of whether any records were skipped initially. If its argument is the keyword **true** (or a boolean expression evaluating to true), it returns the relative record number for the record, counting the first imported record as record number 1. This function returns an integer value.

EXAMPLE:

```
«if recordnumber(true) mod 10 = 0»¶
«endif¶¶                                     Inserts a blank line every ten records.
```

**subfield**

ABBREVIATION:

**sfld**

SYNTAX:

**subfield** *number of expression*

USE:

Returns the indexed subfield from the indicated field or expression; subfield indices start at 1. The expression in *number* must be an integer expression denoting the subfield index. Use with repeating (multivalued) fields. If the specified subfield does not exist, **subfield** returns an empty string.

EXAMPLE:

```
«if sfld 2 of item_num is not empty¶
«subfield 1 of quantity»«endif¶¶
```

**trim**

SYNTAX:

**trim**(*expr1*[ ,*expr2* ])

USE:

Removes any characters in *expr2* from the start and end of *expr1*. If *expr2* is omitted, it defaults to whitespace (spaces and tabs).

EXAMPLE:

```
trim(comment)
trim(comment," !")
```

**word**

SYNTAX:

**word** *start [to end] of expression*

USE:

Returns the indicated word(s) from *expression*. The expressions *start* and *end* must be integer word indices (which start at 1).

EXAMPLE:

```
«word 2 of lname»
```

**wordcase**

SYNTAX:

**wordcase**(*string*)

USE:

Converts a character string expression so that only the first letter of each word is capitalized.

EXAMPLE:

```
wordcase(title)
```

**upcase**

SYNTAX:

**upcase**(*string*)

USE:

Converts a character string expression to uppercase.

EXAMPLE:

```
upcase(exclamation)
```

## Built-In Constants and Variables

### **false**

SYNTAX: **false**  
USE: Used when a literal negative Boolean value is desired.  
EXAMPLE: **«if recordnumber(false) mod 10 = 0» ...**

### **guillemetleft**

SYNTAX: **«guillemetleft»**  
USE: Inserts a literal left chevron mark into the formatted records.  
EXAMPLE: **«guillemetleft»**

### **guillemetright**

SYNTAX: **«guillemetright»**  
USE: Inserts a literal right chevron mark into the formatted records.  
EXAMPLE: **«guillemetright»**

### **it**

SYNTAX: **«it»**  
USE: Variable which holds the value entered by the user in response to the **ask** statement.  
EXAMPLE: **«ask "Enter part number:"» «it»  
«if it is not empty» «it» «endif»**

### **return**

SYNTAX: **return**  
USE: Inserts an ASCII carriage return into an expression.  
EXAMPLE: **«line 1 of a & return & line 2 of a»**

### **quote**

SYNTAX: **quote**  
USE: Inserts a literal quotation mark into an expression.  
EXAMPLE: **«a && "said" & quote & b & quote & "."»**

### **tab**

SYNTAX: **tab**  
USE: Inserts an ASCII tab into an expression.  
EXAMPLE: **«a & tab & b»**

### **true**

SYNTAX: **true**  
USE: Used when a literal positive Boolean value is desired.  
EXAMPLE: **«if recordnumber(true) mod 10 = 0» ...**

## Integer Operations

The following operators may be used in constructing integer expressions. All connect two integer operands and return an integer. All operators take two arguments, using the syntax *i1 op i2*, where *i1* and *i2* are integers or integer expressions, and **op** is one of the operators listed below (e.g. **counter + (a / 2)**).

<b>+</b>	addition
<b>-</b>	subtraction
<b>*</b>	multiplication
<b>/</b>	division: returns an integer, discarding any remainder (e.g. <b>12 / 5 = 2</b> ).
<b>mod</b>	modulus operator: the remainder from dividing the left operand by the right operand. Examples: <b>14 mod 5 = 4</b> and <b>20 mod 3 = 3</b> .

Take care to avoid divide-by-zero errors with **/** and **mod**.

## Comparison Operations

The following operators may be used in constructing logical (boolean) expressions. All connect two operands and return a boolean value. All operators take two arguments, using the syntax *e1 op e2*, where *e1* and *e2* are expressions, and **op** is one of the operators listed below (e.g.: **price <= 10, last = "Smith"**).

Each comparison operator first attempts to compare the two items as numeric quantities; if it cannot do so (because either is not a well-formed number), then the two items are compared on a character by character basis as strings (as in HyperTalk). Comparisons of character strings are performed based on the strings' relative alphabetical order.

<b>&lt;</b>	less than
<b>&gt;</b>	greater than
<b>=, is</b>	equals
<b>is not, &lt;&gt;</b>	not equal (the not equal sign is also equivalent)
<b>&gt;=</b>	greater than or equal to (the single greater-than-or-equal sign is also equivalent).
<b>&lt;=</b>	less than or equal to (the single less-than-or-equal sign is also equivalent).
<b>is empty, = ""</b>	is empty (contains no characters)
<b>is not empty</b>	is not empty (contains at least one character); note that the form <b>if fieldname</b> is equivalent to <b>if fieldname is not empty</b> . A not equal sign followed by two double quotation marks is also equivalent to <b>is not empty</b> .
<b>is a[n] type</b>	Tests whether the first operand is of the data type specified in <i>type</i> (one of <b>number</b> , <b>logical</b> and <b>integer</b> ).



## Logical Operations

- and** Joins two conditions (boolean expressions) such that the compound condition is true only if both conditions are true. Example: «if **Iname is not empty and fname is not empty**» «Iname» «fname» «endif»
- or** Joins two conditions (boolean expressions) such that the compound condition is true if either or both conditions are true.  
Example: «if **Iname is "Smith" or Iname is "Jones"**» «next» «endif»  
*Skips records for Smiths and Joneses.*
- not** Logically negates its argument.  
Example: «if **not comment contains "."**»...«endif».

## Grouping Operations

- ()** Parentheses may be used to specify the evaluation order for expressions involving more than two logical conditions or arithmetic/string operations: «if (**Iname is "Smith" or Iname is "Jones"**) and **fname is "John"**»...«endif» or «put **(a-b)/c** into sum».

## InData Technical Information

### InData Limitations

In general, the only limitations imposed by InData are those resulting from primary memory limitations. There are no built-in limits on the number or length of fields and records nor on the size (length) of a prototype.

### InData Memory Usage Under Mac OS 9

InData lets you push InDesign far beyond where most users can take it manually. For example, a user interactively building a complex document interactively would give up long before InDesign started taking three minutes to insert a new page because of inadequate memory allocation, but InData will blindly go far beyond this limit, if asked. And, InData has no way of directly judging how much memory you need for a given import, so it can't complain intelligently when memory gets "too low."

The solution is simply put, but harder to achieve: you must allocate adequate memory for InDesign, using the Finder's **Get Info...** dialog, if you want reliable operation with InData. There's no magic setting of InDesign's memory allocation that's adequate for all situations, but, luckily, the Finder's **About this Computer...** dialog provides an excellent graphical memory utilization monitor for InDesign (and any other application). When using InData, keep this memory monitor dialog open, and if you see InDesign using all or nearly all its allocated memory during or after an InData import, allocate InDesign more memory (if possible) until there's some free space available during InData import.

These considerations do not apply to Mac OS X nor to any Windows operating system which implements virtual memory management.

## InData Reserved Words

The following words (identifiers) are reserved as keywords in the InData prototype language, and cannot be used for any other purpose (in alphabetical order, with abbreviations in parentheses):

**a, an, and, ask, character (char), columns, contains, div, down, else (|), end, endif (|, fi), exit, fields, forever, hidden, if (|), in, into, is, item, line, marked, mod, next, non-styled, not, of, once, open, or, previous (prev), put, quoted, read, recordfields, repeat, set, styled, subfield (sfld), the, then, this, to, unquoted, unstyled, until, while, with, word.**

The following identifiers are reserved as built-in constants, global or local property names, and built-in functions in the InData prototype language, and cannot be used as field names or variable names (in alphabetical order, with abbreviations in parentheses):

**aspectratiofit, autostart, center, charonum, clipboard, debugterse, debugtree, dec2frac, defaultisstyled, downcase, empty, endrecord, false, fieldquote, fieldseparator (fldsep), fieldindex, fieldvalue, filecontents, fileexists, filename (fn), filequoting, fileinfo, filetype, firstmaster, fit, fithframev, framefit, guillemetleft, guillemetright, ignoremissingpictures, integer, it, itemdelimiter, length, logical, master, number, numtochar, offset, page, pageinfo, picture (pic), picturefolders, pictureimportcrop, pictureimportpage, pictureposition (picpos), quote, randomfonts, recordnumber, recordseparator (recsep), return, sequentialfonts, spread, startrecord, subfieldseparator (subfldsep), tab, topleft, trim, true, upcase, viewfrequency, wordcharacters, worddelimiters.**

## Additional Prototype Restrictions

The built-in field names—**a** through **z**—cannot be reassigned. Thus, the following **fields** statement is illegal, because it attempts to reorder the built-in fields:

```
«fields b,a,d,c,f,e|»
```

InData statements between chevrons (*including* the chevron marks or the chevron mark and the terminating paragraph mark) must be formatted in a single character style; changing any aspect of the character style in the middle of a statement is not allowed, including kerning. Thus, the following prototype statements *are all illegal* (look closely!):

```
«if last is empty»
«lname »
«else»
«la st|»
```

## **Processing Very Large Data Files**

The practical limits on the number of pages that InDesign can handle within a single document is in the many hundreds of page range (although InDesign will need quite a bit of memory). Therefore, if you have a very large number of records and are placing only a few records per page, you'll have to process the data file in sections to avoid exceeding some practical maximum. We recommend keeping documents to a maximum of a few hundred pages so that they do not become unwieldy. This is not a hard limit, of course, but only a recommendation, and depends on the speed and size of your system.

## **Page Complexity**

We recommend keeping the number of linked text frames per page to a minimum, especially if the number of document pages is large, or if you're importing graphics along with text.



# 14

## Troubleshooting and Error Messages

This chapter discusses some common problems encountered using InData, and presents solutions to them. It also lists and discusses all of InData's status and error messages.

### Common Problems and Their Causes

**There is no InData menu in the InDesign top-level menu.**

This usually means that InData is not installed correctly. InData must be placed in the Plug-ins sub-folder (or one of its sub-folders), and InDesign (re)started.

**I get only one page or one spread of formatted data. The rest of my data disappeared.**

(You should have received a warning from InData that the data overflowed.) The problem is usually that you're not importing into an automatic text thread, so InDesign isn't creating extra pages as needed.

- ◆ Make sure you have installed the InFlow plug-in along with InData.
- ◆ Make sure you're using text frames created on the master pages, not frames you've created manually in the document pages (except in very rare circumstances).
- ◆ Make sure the text frames you're using from the master page are linked into the automatic text chain (see chapter 4).

An easy way to test for this problem, before importing, is to place the insertion point at the end of the prototype, and type Shift-Keypad Enter (the new frame character) several times. If InDesign doesn't create at least one new page or spread when you do this, then you're not importing into a text frame that's part of the automatic text flow.

Incorrect use of the enhanced **Keep with Next** and **Keep Lines Together** paragraph attributes. If you have used these settings in your prototype, try the following:

- ◆ Select all of the text in the text frame.

- ◆ Select **Style=>Formats...** and disable both the **Keep with Next** and **Keep Lines Together** paragraph attributes.

If this procedure corrects your problem, then you have been too enthusiastic in applying these paragraph attributes, and you'll need to give InDesign some permissible break points in your prototype.

**There are extra lines inserted into each record that I don't want.**

This problem is usually caused by one of the following.

- ◆ There are empty data fields in some records. In this case, the extra lines won't be present for all records, just a subset of them. This can be corrected by using conditional statements for those data fields.
- ◆ There are paragraph marks left outside of an **if** statement that should be inside of it. If you only want a new paragraph to begin if a condition is met (or not met), then be sure to place the paragraph marker *between* the **if** and the **endif** statements, where you need it.
- ◆ A prototype statement is closed with a right chevron and is also followed by a new paragraph mark or line break—often, the **fields** statement is the culprit. Either a chevron or a paragraph mark may be used to mark the end of an InData prototype statement, but not both—when you use both, the paragraph mark is taken literally and included in the formatted data, giving you the blank line.

**The records of my data are running together.**

This is the opposite of the previous problem. This usually is caused by a missing paragraph mark at the beginning or end of the prototype, or by including such a mark inside a conditional, when it should be outside.

**I want my records to be on separate pages (or frames), but they're running together.**

This is usually caused by a missing new frame or new page character at the end of the prototype.

**All or most of my fields are missing, or are all bunched up in one place.**

You've probably chosen the wrong basic file format, i.e., comma-separated instead of tab-separated, or vice versa. Verify that this is the problem by looking at the data, and use the other format.

**One (or more) of my fields is in the wrong place.**

It's very likely that the order of the fields declared in the prototype's **fields** statement doesn't match the order of the fields as exported from your database or spreadsheet.

One way to verify this would be to temporarily add one line at the start of the prototype for each field, with each field placeholder prefixed literally by its field name. For example:

```
«fields partnumber, price, description»  
partnumber: «partnumber»  
price: «price»  
description: «description»  
...
```

The results of importing your data should make it clear if there's a mismatch between your declared field order and the actual data's field order, based on each field's expected and actual contents. E.g., if the result of importing the example above were

```
partnumber: 3.25  
price: 20222  
description: 10 1/2" threaded bolts
```

you'd know that the **partnumber** and **price** fields were probably reversed.

In any case, to solve this problem, either change the order of the exported fields in the original application and re-export the data, if appropriate, or change the order of the fields in your prototype's **fields** statement to match the actual order of the data fields.

#### **One or more field names are appearing literally in my output.**

You've probably forgotten to declare the field name in a fields statement, or have misspelled a field name, or have selected just a part of the prototype—rather than just placing the cursor in the prototype, resulting in an insertion point—prior to importing the data.

## Status Messages

As InData does its work, it displays various messages in the status area of its control panel. This section lists those messages and their meaning, roughly in order of appearance. (Some of these messages flash by so quickly you'll never see them except in particular circumstances, but they're all listed here for completeness.)

#### **Initializing...**

InData is just starting up and initializing itself.

#### **Scanning prototype...**

InData is examining the prototype you gave it, and preparing an internal ("executable") version for later use during data import. Structural errors in the prototype are found during this phase of operation, as are errors in **fields**, **open**, and global **set** statements.

#### **Please select a data file to import.**

InData is waiting for you to select a file to import, from its file selection dialog.

#### **Opening file...**

InData is opening the data file that you specified.

**Initializing for data access...**

InData is preparing to import data from the now successfully-opened data file.

**Ready to import data from file “\_\_\_”.****Ready to import data from clipboard.**

InData is waiting for you to select any special options in its control panel before pressing either the **Start** button to start data importation, or the **Cancel** button to abandon this data import immediately (with no side-effects).

**Auto-starting...**

The auto-start option has been selected in the InData **General Preferences** dialog, or InData has seen an «**set autostart to true**» statement in the prototype, and is beginning data importation without waiting for you to press the **Start** button.

**Paused.**

InData has temporarily paused data import, after updating the document view to show its progress to this point. It is waiting for you to press either the **Continue** button to go on with the data import from where it left off, or the **Stop** button to stop data import immediately.

**Skipping initial data from file “\_\_\_”...****Skipping initial data from clipboard...**

InData is skipping any data records before the first record to be imported. During this time, the progress bar is a lighter shade of gray than it is during normal data importation.

**Importing data from file “\_\_\_”...****Importing data from clipboard...**

InData is now importing data from the named file or from the system clipboard. The dark gray bar below this status message shows the progress InData has made. The length of this progress bar relative to the total length of the measure below it shows the current position in the data file relative to the total length of the data file (if there is no specific end record), or shows the current record number relative to the end record number (if there is a specific end record).

**Waiting for InDesign to update document view...**

InData is now waiting for InDesign to compute an updated view of any imported data. This may take a significant amount of time after a large amount of data was imported, particularly if you didn't select view updating during importation.

**Waiting for page headers/footers update...**

InData is now waiting for its automatic page header/footer update to finish, after finishing an import employing marked text.

**Finishing... or****Cleaning up... or****Cancelling... or****Stopping...**

These all mean the same thing—InData is now shutting down—but reflect various ways in which InData can stop. Respectively, InData is finishing normally,



cleaning up after one or more fatal errors, cancelling before data importation began (because you pressed the **Cancel** button), or stopping during data importation (because you pressed the **Stop** button).

**Finished.** *or*

**Failed.** *or*

**Cancelled.** *or*

**Stopped.**

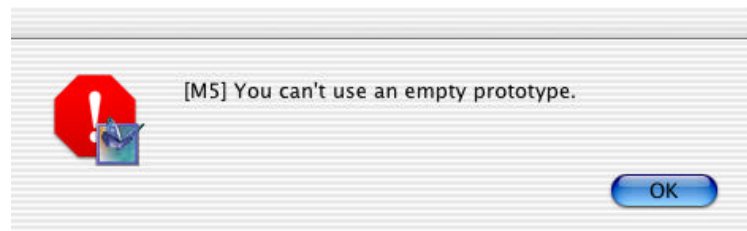
**Imported \_\_\_ records per minute.**

InData has finished in some fashion, and is informing you of how many records per minute it imported, on the average. Respectively, data import has finished normally, has stopped because of a fatal error, has been cancelled with the **Cancel** button, or has been stopped with the **Stop** button.

This message, along with the control panel window, will stay up for several seconds after finishing, to let you read it (unless you've cancelled, in which case it will disappear almost immediately). If you want to see it longer than that, simply click in the title bar of the InData control panel window, and hold down the mouse button—the control panel will stay up as long as you hold down the button.

## Error Messages

InData will put up an alert dialog when it encounters an error, requiring you to confirm that you've seen the error message by pressing the **OK** button (or by typing a carriage return). Here is an example of what an InData error alert looks like:



For file opening errors, InData will produce a standard InDesign-style error alert (e.g., **File is already open for writing**).

After fatal errors, InData will exit, returning you to InDesign, after first displaying the final status message **Failed** along with any import speed information (discussed previously). After non-fatal errors, you can correct the situation that caused the error and retry—e.g., you would correct an improperly formed starting record number and press **OK** again, in the **Range Preferences** dialog.

An explanation and, in some cases, a suggested cure follows each error message. In these explanations, a “\_\_\_” placeholder in the message stands for an addition-

al piece of specific information about the error (for example, the name of the offending field in an error message about an invalid re-definition of a field).

There are four “phases” of InData operation during which errors may be discovered:

- ◆ User interface handling (e.g., dialog interaction);
- ◆ Prototype structural analysis (right after you select **Import from ...**);
- ◆ One-shot prototype processing (“execution”), right after prototype structural analysis—at which time **fields**, global property **set**, and file **open** statements are handled; and
- ◆ Normal prototype processing (“execution”) during data import.

The latter two phases are lumped together below, but errors discovered during one-shot prototype processing are reported before anything else happens (such as getting a dialog for the file to import).

On the Macintosh, any error that refers to “not enough memory” must be dealt with in one of two ways. Either try to reduce InDesign’s memory requirements in some way—close all unneeded documents, or move all unused plug-ins and filters out of the InDesign folder and restart InDesign—or else try to increase the amount of memory the system gives to InDesign (exit the current InDesign session, select the InDesign application in the Finder, invoke **Get Info**, increase the current size field in the Memory section of the resulting “InDesign Info” window, and then restart InDesign with the new memory allocation).

Any error that involves a malformed dialog field leaves the offending field fully selected. Thus, in the simplest case, you can just re-type the field in error and try again.

## User Interface Errors

**[M3] InData can’t initialize itself properly; most likely, there’s not enough memory available.**

**[M8] InData can’t initialize itself for data import; most likely, there’s not enough memory available.**

InData is unable to get started in various ways, due to lack of memory (or, less likely, due to something even more drastic).

**[M6] You can’t have more than one source text frame candidate on the current spread’s pasteboard.**

**[M32] You must have a source text frame candidate on the current spread’s pasteboard.**

You’re using **Import from Pasteboard...**, and InData is looking for a text frame to use as the data source for the import, but either you have more than one text frame on the pasteboard of the current spread, or you have no text frames on the

pasteboard, respectively. (Note that all frames in any prototype story chain on the pasteboard are ignored for the purposes of finding the data source text frame.)

**[M9] You can't use a non-digit in this character specification.**

One of the character specification fields in the Data Format dialog is malformed. These errors will only occur if the field contains more than one character—if it contains just one character, that character is taken literally as the specification—in which case the characters are interpreted as a decimal value for a Unicode character. If you have a character that's not a digit (0 to 9), fix it.

**[M11] The first record to import isn't a valid positive number.**

**[M12] The last record to import isn't a valid, positive number, nor is it one of 'last', 'end', 'all', or 'final'.**

**[M13] You can't use a last record that's less than the first record to import.**

These are self-explanatory problems with one of the two fields in the **Range Preferences** dialog.

**[M1] Uh-oh, the self-consistency check named “\_\_\_” has failed. Please report the exact circumstances to Em Software.**

**[M14] InData unexpectedly failed to load its preferences.**

**[M15] InData unexpectedly failed to save its new preferences.**

These errors will occur only when something is wrong with InData internals—nothing you can do should be able to cause them, nor can you fix them, though perhaps you can find some way to work around the problem. Contact Em Software if you receive one of these error alerts. (We will most likely need an electronic copy of your prototype document and data file to fix the problem.)

**[M17] The number of records between view updates isn't a valid, positive number.**

This is a self-explanatory problem with the single editable field in the **View Preferences** dialog.

**[M25] Warning: Your imported text overflowed its text chain. Unless you're doing something special, check to make sure you're importing into the automatic text frame.**

You'll receive this message after all data import is finished, if InData detects that your data has overflowed its containing story. See the description of the common problem, “I get only one page or spread of formatted data,” at the start of this chapter.

**[M26] You're importing into a story that jumps from one page to an earlier page at some point; this prevents automatic header/footer update.**

You'll receive this message when InData is about to update your document headers and footers (whether automatically or manually), and it discovers that the current story has backward links in the document. (This can only happen if you've manually linked some text frames.) InData's header/footer update machinery can't deal with this situation, so you'll have to rearrange your text links to accommodate InData's limitation.

**[M33] Sorry, that name is already taken; choose another.**

Given in the **Name Story...** or **Name Substory...** dialog when a given name is already used in a document.

**[M34] Sorry, that name is too long; use a shorter one.**

Given in a **Name Story...** dialog when a given name is too long (currently, longer than 63 characters).

**[M46] Named story not found.**

Given in the **Find Story/Substory...** dialog when the named story wasn't found in the current document.

**[M47] Named substory not found in named story.**

Given in the **Find Story/Substory...** dialog when the named substory wasn't found in the named story.

**[M48] The maximum variable length isn't a valid, positive number.**

Given in the **Preferences=>General...** dialog when the given maximum variable length (in Kbytes) isn't a valid number greater than zero.

**[M49] Sorry, that continued string is too long; use a shorter one.**

Given in the **Make Header/Footer...** dialog when the given **Continued** string is too long (currently, longer than 63 characters).

**[M50] The number of lines isn't a valid, positive number between 1 and 99.**

Given in the **Maker Header/Footer...** dialog when the given number of lines to search is not a valid number in the range 1 to 99.

## Prototype Structure Errors

For errors discovered during the prototype structural analysis phase, InData will either select the discrete offending portion of the prototype, or else leave the insertion point where the error was detected (which often isn't where the error actually occurs). For most of these errors, the general solution is to "fix the error and try again."

**[M5] You can't use an empty prototype.**

You invoked InData with an empty prototype (either in the current text frame, or in the story designated by **Use Story as Prototype**). You must supply a non-empty prototype.

**[T1] You can't have this extra decimal point in a number.**

**[T2] You can't use this character in a number.**

InData has found something wrong with a number in your prototype. You've probably forgotten a space between two elements, or simply made a typing error.

**[T3] You can't use this decimal point in an identifier.**

InData found a decimal point (period) in an identifier, which can't contain such a character.

**[T4] This string constant needs a closing double-quote. (It continues past the end of line.)**

**[T5] This string constant needs a closing double-quote. (It continues into a closing delimiter.)**

You've forgotten to close a string constant, or else are trying to use a double-quote character in a string constant (which you can't). The following two examples illustrate these errors, respectively. The solution is to close the first kind of string, and avoid the second.

```
«put styled "<B>An unterminated bold phrase»  
«if a = "a double-quote " character¶
```

**[T6] This prototype is too large for available memory.**

**[P16] This prototype is too large for available memory.**

Both of these messages mean the same thing: you're trying to use a prototype which is too large for InData to convert into its internal ("executable") form, given the amount of available memory.

You may also be mistakenly using many pages of already-imported data for your prototype. This can happen if you try to import after another import has finished, without first restoring the original prototype. (That's one reason we recommend you use a separate prototype on the pasteboard.)

If you're really trying to use a large prototype, and get this message, then increase the amount of memory available to InData.

**[T7] You can't use this character in a prototype.**

You're using some strange character in an InData prototype statement (inside chevrons) that InData doesn't know how to handle. You may want to delete it, or move it outside the chevrons to place it literally in your formatted data.

**[T8] This identifier or constant is too long.**

You're using an identifier (field, variable, property, or constant name) or a string constant of length greater than 1,024 characters. Use a shorter name or constant.

**[P1] You can't change character styles here, inside an InData statement.**

You're using an InData prototype statement whose character styles aren't totally consistent. Note that the prototype statement, for this purpose, includes both the opening chevron («) and the closing chevron (») or the closing new paragraph (or new frame, new column, or new page) character.

This is one of the easiest errors to make, and one of the hardest to fix at times, because you can't always see what's inconsistent. InData will leave the insertion point before the first character that's "different," so you only have to figure out what character style element is different, and change it to match the rest of the statement. An easy way to fix this problem is to select the entire statement, including both the opening and closing delimiters (whether chevrons or paragraph characters), open the **Character** palette from InDesign (with the

**Text=>Character...** menu selection), change all the settings to what you want, leaving no setting indeterminate, and then press **OK**.

Note that you can't apply manual kerning to any portion of an InData prototype statement, even between a closing chevron and the following character (which would be a logical place to kern—this is a limitation we hope to remove someday).

**[P2] You're using this closing chevron (») without first using an open chevron («).**

**[P3] You're using an open chevron («) somewhere without a matching close chevron (»).**

Both of these errors reflect a mismatch of prototype statement delimiters. Add the required chevron or new paragraph at the appropriate point.

If you need to use a chevron character in your formatted data, then you can use the constructs **«guillemetleft»** to insert a « and **«guillemetright»** to insert a » (guillemetleft and guillemetright are the Adobe Postscript character set names for the chevrons).

**[P17] InData expected to find \_\_\_, but found the selected text instead (or ran into the end of the prototype).**

This is a “kitchen sink” error message, used when the InData structural analyzer didn't find what it expected. It's not terribly helpful, because what it expected to find often has only an obscure relation to what you're trying to do, and where it found the error often has nothing to do with the real location of the problem. (Computer language parsers still aren't very well human-engineered, nor are they willing to forgive even the slightest error.)

Perhaps the most common source of this error (and the most misleading resulting error message) is forgetting to close an **«if»** statement with an **«endif»**: InData leaves the insertion point at the end of the prototype, because the error won't be spotted until then, but that's not normally where the **«endif»** should go.

Another common source of this error is mistyping a keyword such as **fields**, **if**, etc., or forgetting a comma between field names in a **fields** statement.

**[P19] Only anchored picture frames are supported.**

You've included an anchored text frame in your prototype. Presently, InData only knows how to deal with anchored graphics frames. Either delete the anchored text frame, or replace it with an anchored graphics frame, if that's what you meant to include.

**[P21] You can't use more than one of 'styled', 'marked' and 'into' in this 'put' statement.**

You're trying to use conflicting forms of the put statement simultaneously. Decide which one you want, and get rid of other(s).

**[P25] You're giving too many arguments to this function call.**

You're using more than two arguments to a function call, which is beyond the current limit. (No InData function takes more than two arguments.) Get rid of the extra argument(s).

**Prototype Execution Errors**

For errors encountered during data importing ("prototype execution"), InData usually appends the following information to the error message: **(This error occurred while importing record number \_\_\_\_.)** This knowledge may help you track down the error, particularly if it's data-related—you can examine the data record in question with a text editor, and see what might be causing the error.

Many of these messages begin **You can't...** This isn't meant literally, since many errors can be triggered quite indirectly, by operating on malformed data far, far away from your original prototype intent—it's just a short way of saying "there's a problem here of the following nature."

Any error message beginning **Warning:** is not a fatal error, but is rather a problem that you should know about (it usually involves the potential loss of data, such as missing pictures, or overflowed text framees).

**[M4] You can't import any more records in this demonstration version.**

You're limited to 30 records in the demonstration version of InData, and you're trying to exceed that limit. Import fewer records. (You should never see this in the full product.)

**[E1] You can't use a string (in this case, beginning '\_\_\_\_') longer than 32,767 character in a string comparison test, or in a 'is in', 'is not in' or 'contains' test.**

One of your fields contains more than 32 KB of data (which is fine, in itself—InData can read and place fields of any length), and you're trying to use it in a string comparison or string containment test of some sort. You'll just have to avoid using this field in this test, for this set of data. (This is a very unlikely error.)

**[E4] You can't 'set' a field (in this case, '\_\_\_\_').****[E39] You can't 'set' a variable (in this case, '\_\_\_\_').**

You're using a field or variable as the target of a **set** statement, which makes no sense (the **set** statement is for setting properties, not variables or fields). Use the **put** statement to change a variable; you can't change a field.

**[E5] You can't have a 'fields' statement inside a conditional.****[E24] You can't set the global property '\_\_\_\_' inside a conditional.**

All **fields**, global **set**, and **open** statements must be at "top level," that is, not inside an **if** statement. E.g., the following **fields** declaration is not at top level: **«if field»«fields name, address»«endif»**. Move the offending statement to near the start of the prototype, outside of any **if** statement.

**[E6] You can't re-use the field name '\_\_\_', which is already in use (as a different field).** You're using the given field name in a contradictory fashion. E.g., in the prototype fragment:

```
«fields name, address, zip
«fields name, zip, phone
```

the field named **zip** is being declared first as field number three, and subsequently as field number two. Fix the conflicting **fields** statement.

**[E7] You can't use '\_\_\_' as a field name, since it is already in use as a constant, variable, or function name.**

The given field name is being used for some other purpose, so you'll have to choose another field name for use in your prototype. (Be sure to change all occurrences of the offending field name—it's easy to forget one or two.)

**[E8] InData can't handle all the fields being named in a 'fields' statement, because there's not enough memory available.**

You'll have to increase the memory available to InData, somehow, or use fewer field names. (Note that there's no built-in limit on the number of fields—it's just a question of having enough memory available. There are real-life cases of 600 fields in a prototype.)

**[E9] InData encountered an I/O error reading the data file.**

InData was told by the host operating system that something went wrong, whilst reading from your source data file. This is probably most commonly caused by a file server shutting down while you're importing data from a file living on that server, or by bad media on a hard or a floppy disk drive. You'll have to consult higher authorities for this particular error.

**[E11] InData can't store the current input record, because it's too large for available memory.**

Either you're running low on available memory, or else you're trying to read a large record from your data file. In either case, you'll have to give InData (InDesign) more memory to work with. (InData has no built-in limit on the size of its input records—it's been tested with input records of multiple megabytes.)

Most likely, there's a stray double quote character in an input field that's causing InData to attempt to read a huge amount of data until the next double quote.

**[E12] You can't use a non-constant expression (in this case, involving '\_\_\_') in an 'open' or a global 'set' statement.**

You're trying to use a field or a variable in a context where no variability is permitted (during the "one-shot" prototype processing). E.g., **«set picpos to a»** will cause this error, since the value of field **a** is undefined when setting the global picture position property before starting import.



**[E13] You can't use the value '\_\_\_' as a conditional (Boolean) value; it should be either 'true' or 'false'.**

Usually, this error means you're giving an **if** statement a conditional value that's really a string or a number; e.g.:

```
«if char 1 of a» «a» «endif»
```

which isn't what you meant; instead, use something like:

```
«if char 1 of a is not empty» «a» «endif»
```

Another easy mistake is to mentally extend the shortcut **«if field»** to the logical junction operators: e.g., **«if field1 and field2»**, meaning if both **field1** is not empty and **field2** is not empty; unfortunately, this won't work: you must use the full form **«if field1 is not empty and field2 is not empty»**.

**[E14] You can't use a string (in this case, the string beginning '\_\_\_') longer than 255 characters in the current context.**

There are some contexts where InData limits your string values to a maximum of 255 characters—usually, when dealing with file names in the **open** and **set file-name of picture** statements. For example, you may be using picture file names from a field in your data file, and it's certainly possible to have fields longer than 255 characters; if you use a statement like **«set the fn of picture 1 to longname»**, you may receive this error. The only solution is to use a shorter string.

**[E15] You can't use a number that's empty.**

**[E17] You can't use a number (in this case, '\_\_\_') containing more than one decimal point.**

**[E18] You can't use a number (in this case, '\_\_\_') containing an embedded minus sign.**

**[E19] You can't use a number (in this case, '\_\_\_') containing a minus sign but no digits.**

**[E20] You can't use a number (in this case, '\_\_\_') containing embedded spaces.**

**[E21] You can't use a number (in this case, '\_\_\_') containing non-numeric characters.**

**[E31] You can't use a number (in this case, '\_\_\_') that contains more than one minus sign.**

All these errors are simply complaining about your use of a number that's malformed in some way. For example, in the fragment **«if (a + 1) > 10»**, if the field **a** contains the string "1a2", then you'll receive an error complaining about non-numeric characters. Unless you've just made a simple error in your prototype, the only real solution is to make sure the original field in the database is always a valid number, before using it as such in your prototype by testing it with **is a number** or **is an integer**.

**[E16] You can't use a non-integral number (in this case, '\_\_\_') in the current context.**

Some contexts, such as the operands to arithmetic operators, and the indices in **character**, **line**, **item** and **word** operations, require an integral numeric value. The only solution (similar to the errors immediately above) is to make sure you're only using integral values in these contexts. E.g., in general, you can't add or subtract prices in InData (**price1 + price2**), since prices often contain fractional values.

**[E25] You can't set a property of picture '\_\_\_', because no such picture is present in the prototype.**

You're using a statement of the form «**set property of picture *n* to ...**», but there's no *n*th picture in the original prototype. Pictures are indexed starting at 1, from the start of the prototype forward (ignoring the effects of conditionals at import time). Change *n* to correspond to one of the pictures in the prototype.

**[E27] You can't 'set' the unknown property '\_\_\_'.**

You're using a statement of the form «**set property of ... to *expression***», but the given property isn't one that InData recognizes. It's probably a simple typographical error.

**[E28] You can't 'set' a property of an unknown object type (in this case, '\_\_\_').**

You're using a statement of the form «**set property of *objecttype* to ...**», but the given *objecttype* isn't one that InData recognizes (currently, only **picture** is supported as an object type). You've probably made a simple typographical error.

**[E29] Warning: during data import, a picture file was not found, and its picture frame was left empty.**

**[E30] Warning: during data import, \_\_\_ picture files were not found, and the corresponding picture framees were left empty.**

These are fairly self-explanatory. You'll only receive these warnings if you have checked **Keep importing after missing pictures** in the InData **General Preferences** dialog.

If you'd like to know exactly which picture files are missing without going through the entire document, re-do the import, after turning off **Keep importing after missing pictures**, and InData will stop importing at the first missing picture, and tell you its file name. Fix that problem, and import once again, finding the next missing picture; etc.

**[E32] You can't import the picture file '\_\_\_', because it doesn't contain a picture file format that InDesign understands.**

InData found the given picture file, but InDesign can't import it because it doesn't contain a picture type that InDesign can process. Either fix the file name to reference a valid picture file, or change the file to a type that InDesign can import with its **Get Picture...** command.

**[E33] You can't import the picture file '\_\_\_', because the file can't be found.**

InData can't find the given picture file. Check to make sure the file exists, and, if it does, check that the list of picture folders established by any «**set picturefolders to ...**» statement is correctly constructed.

Alternatively, this error message will occur when you are trying to import into a document which has never been saved to disk, and thus has no "home folder"—either because it was created from scratch, generated from a template, or converted from an earlier version of InDesign or from a different platform—regardless of the format of the specified picture file.

**[E34] You can't 'put hidden' the data beginning '\_\_\_', because it's too long.**

InData has a quite generous but fixed upper bound on the length of the data that can be **put hidden**. You'll have to shorten the data you're trying to insert.

**[E38] You're calling an unknown function '\_\_\_'.**

You're calling a function whose name InData doesn't recognize. It's probably a simple typographical error, e.g., «**if** **length(a) > 3**»...«**endif**».

**[E40] You can't put a string longer this long into a variable (in this case, '\_\_\_').**

You're using a statement of the form «**put expression into variable**», but expression resulted in a string longer than the maximum specified in the **Maximum length of variables** field in the **General Preferences** dialog, and you can't store it in a variable. You'll have to shorten the data you're trying to store or increase the maximum variable length setting.

**[E41] You're trying to divide by zero.**

One of the division operators in the prototype detected an attempt to divide something by zero, which is meaningless. You can guard against this case by always testing that a given divisor is non-zero; e.g.,

```
«if a <> 0»«put c / a into quo»«else»«put 0 into quo»«endif»
```

This error can also be triggered by the **mod** operator.

**[E42] You've passed the wrong number of arguments to the function '\_\_\_'.**

You're calling the given function with either too few or too many arguments (operands). Check the InData Reference summary for the correct calling form.

**[E43] You can't create a string (in this case, beginning '\_\_\_') this long with concatenation.**

A string concatenation failed because the resulting string would be larger than maximum variable length given in the **Preferences=>General...** dialog.

**[E44] You can't return a string this long from the function '\_\_\_'.**

A function return failed because the resulting string would be larger than the maximum variable length given in the **Preferences=>General...** dialog.

**[E45] You can't set the property '\_\_\_' to a string this long.**

A **set** failed because the value being set would be larger than the maximum variable length given in the **Preferences=>General...** dialog.

**[E46] You can't 'put ... into' a field (in this case, '\_\_\_').**

A prototype is trying to put a value into a field, which isn't possible, as fields are read-only.

**[E47] You can't get the contents of file '\_\_\_', because it can't be found.**

A **filecontents**(\_\_\_) failed, because the named file wasn't found.

**[E48] InData encountered an I/O error reading the file '\_\_\_'.**

A `filecontents(___)` failed, because it received an error while reading the file contents.

**[E50] The InCatalog link key value beginning '\_\_\_' is too long.**

The specified InCatalog link key value is too long and must be shortened.

## AppleEvent Scripting Errors

The following errors are all scripting-related, returned as an error parameter by a failing `import data` AppleEvent.

**[M35] Input file specification (direct parameter) missing or invalid.**

No input file parameter was given in the `import data` AppleEvent.

**[M36] Prototype story name parameter missing.**

No prototype story name was given in the `import data` AppleEvent.

**[M37] Named prototype story not found.**

The prototype story name given in the `import data` AppleEvent wasn't found in the current document.

**[M38] Target story name parameter missing.**

No target story name was given in the `import data` AppleEvent.

**[M39] Named target story not found.**

The target story named in the `import data` AppleEvent wasn't found in the current document.

**[M40] Input file not found or not openable.**

The input file given in the `import data` AppleEvent wasn't found, or wasn't openable.

**[M41] No document is currently open.**

An `import data` AppleEvent was given when no document was open.

**[M42] Named substory in target story not found.**

The target substory named by the `import data` AppleEvent wasn't found in the named target story.

# Index

## SYMBOLS

- 93  
-- 85  
& 107  
&& 107  
( ) 93  
\* 93  
/ 93  
[ 92  
| 92  
+ 93  
< 88  
<= 88  
<> 88  
= 87  
> 88  
>= 88  
« and » 15, 108

## A

address labels 65  
and operator 92  
AppleScript 45, 49, 153, 156, 200  
    errors 200  
arithmetic operators 93  
ask statement 141, 169  
autoflow 8  
automatic field names 56  
automatic text chain 54  
automating data importing 45  
auxillary folders 9  
avoiding blank lines 29-30, 94

## B

Basic format pop-up 18  
boolean expressions 92

## C

case construction 91  
case conversion functions 112  
case sensitivity 61  
character operator 30-32, 103, 174  
character sets 165  
character to number conversions 112  
chartonum function 112-113, 175  
chevrons 15, 55, 108, 179  
    inserting literal 108  
    typing 23  
    when to omit 33, 63  
clipping paths 121  
comma-delimited format 69, 77  
comment designator 85  
comparing records 33, 94  
comparison operators 87, 180  
complex conditions 92  
    parentheses in 93  
conditional importing 29-30, 85  
contains operator 88  
control panel, see InData control panel  
conversion functions 112  
custom date formats 70

## D

data import procedure 17-18  
    undoing 19  
Data Preferences panel 140  
data validation 150  
database programs 79, 81  
debugging 147  
dec2frac function 112-113, 175  
defaultisstyled setting 143  
defaults 25  
demo version 9  
division by zero 93  
document setup 21, 53

Document View during Import dialog 71, 165  
double hyphen-to-em dash conversion 144  
downcase function 112, 175

## **E**

else if statement 90  
else statement 87  
em-dash 85  
end repeat statement 99  
endif statement 86  
error messages 185-200  
    scripting-related 200  
examples 111, 119, 157  
    detectives 119  
    invoice 111  
    scripting 157-158  
Excel 83  
exit repeat statement 101, 169  
exit statement 135, 169  
exporting data files 78

## **F**

false constant 178  
fi statement 92  
field placeholder 15  
field placeholders 56  
field separator 174  
fieldindex function 115  
fields statement 15, 55, 142, 169  
    multiple 142  
    skipping fields with 62  
fieldvalue function 115  
filecontents function 145, 175  
fileexists function 145, 175  
fileinfo function 175  
FileMaker Pro 79, 111  
Find Story/Substory dialog 165  
fixed-width fields 140  
footers 40, 130  
forcing text to next column/text box 66  
FoxBase/FoxPro-style quoting 70  
frameinfo function 176

## **G**

General Preferences panel 74-75, 122, 166.  
    See , pictures  
    Keep importing after missing pictures  
        checkbox 125  
getting started 11  
graphics. See pictures  
grouping operators 181  
guillemetleft constant 179  
guillemetleft operator 108  
guillemetright constant 179  
guillemetright operator 108

## **H**

header record 70  
headers 40, 130. See , pictures  
hidden keyword 128  
hidden text 155, 170

## **I**

if statement 86, 169  
    alternate forms 92  
    nested 93  
    paragraph marks in 94  
import data AppleEvent 153  
import data format 18  
Imported Data Format dialog 18, 164  
    Basic format pop-up 164  
importing 71  
    controlling 72  
    range of records 71  
InCatalog 200  
InData control panel 68, 72, 163, 187-188  
    messages in 187-188  
    status messages 187-188  
InData menu 53, 67, 130, 162  
    About 68, 163  
    data import selections 57  
    Find Story/Substory 67, 155, 162  
    Import from Clipboard 67, 162  
    Import from File 57, 67, 162  
    Import from Pasteboard 67, 162

- Make Header/Footer 67, 162
- Name Story 67, 162
- Name Substory 67, 155, 162
- Preferences 67, 74, 163
- Preferences submenu 163
- Update Headers/Footers 67, 130, 162
- Use Story as Prototype 57, 67, 162
- InData.Reg file 9
- InDataImportFromFile method 158
- InFlow 8
- inserting files during import 145
- integer operators 180
- interchange formats 77
- is a operator 140
- is empty operator 87
- is in operator 88
- is not empty operator 87
- is not in operator 88
- is not operator 88
- is operator 88
- it constant 179
- it variable 141
- item delimiter 172
- item operator 110, 176

## K

- keyboard equivalents 161
- knock-outs 34

## L

- length function 31-32, 106, 176
- limitations 181
- line operator 109, 176
- logical connectives 92
- logical operators 181
- loops 99
  - exiting from early 102
  - variable indexed 100

## M

- Mac OS 9 vs Mac OS X 1
- Make Header/Footer dialog 42, 129, 167

- manually-created data files 84
- mark 41, 127
- mark reference 42, 127
- marked keyword 41, 127
- master pages 13, 132
- maximum length 75
- memory usage 9
- memory usage under Mac OS 9 181
- menu, see InData menu
- mod operator 93, 180
- modulus 93
- MS Word comma-delimited format 77
- MS Word merge comma-delimited format 69
- MS Works tab-delimited format 69, 77
- multi-part documents 156

## N

- Name Story 155
- Name Story/Substory dialogs 167
- naming data fields 55
- new box character 66
- new column character 66
- new line character 24, 29-31
- next keyword 94
- next operator 177
- next repeat statement 102, 170
- next statement 135, 170
- nonstyled keyword 144
- number of items statement 110
- number of lines statement 110
- number of statements 177
- number of words statement 108
- number to character conversions 112
- numtochar function 112, 177

**O**

offset function 107, 177  
open statement 170  
or operator 92

**P**

page number placeholder 22  
pageinfo function 114, 177  
parentheses 181  
PDF files, pictures from 118  
    cropping 118  
pictures 36, 74, 117-126  
    conditionally importing 38  
    file locations 124  
    frame attributes 121-123  
    missing files for 40, 125  
    PDF file page as 118  
    positioning and sizing 74, 121  
    precise placement of 125  
    scaling 121, 123  
    set pictureposition statement 122  
    setting attributes of 38, 122  
preferences 67, 74, 163  
previous keyword 33, 94, 135  
previous operator 177  
prompting the user 141  
prototype 53, 86, 117, 127, 135, 147, 168, 195-197  
    applying master pages 132  
    prototype 132  
    comparing records in 33  
    conditions in 86  
    debugging 147  
    designating 59  
    downplaying 150  
    error messages 195-197  
    errors in 73  
    inserting file from 145  
    invalid format 26  
    literal text in 15  
    loops in 99  
    mark in 127  
    multiple text boxes for 149

    on pasteboard 59  
    paragraph marks within 29-31  
    picture box in 36, 117  
    placement 14, 58-60  
    prompts from 141  
    resetting 20  
    restrictions 182  
    selecting 60-61  
    setting picture attributes in 122  
    testing 147  
put hidden statement 171  
put marked statement 170  
put statement 75, 127, 137, 170  
    defining variables with 137  
    hidden keyword 128  
    inserting file with 145  
    into keyword 170  
    marked keyword 127  
    nonstyled keyword 144  
    quoted keyword 144  
    styled keyword 143  
    unquoted keyword 144  
    unstyled keyword 144  
put styled statement 171

**Q**

quote constant 179  
quoted keyword 144

**R**

Read Me file 7  
read statement 135, 171  
Record Preferences panel 71, 168  
record separator 174  
recordnumber function 96, 114, 178  
reimporting 57  
remainder 93  
repeat statement 171  
    repeat forever 101  
    repeat until 100  
    repeat while 100  
    repeat with 100  
repeating fields 81, 111



reserved words 182  
retrieving the next data record 135  
return constant 179  
return global constant 144  
reversed type 34, 60  
right chevron mark 33  
rules 34, 98  
    vertical 98  
running headers/footers 40, 127  
    changing after data import 131  
    headers, footers. See also  
    updating 130

## S

samples 7, 54, 91, 120, 140, 150  
    dump fields 150  
    mailing labels 54  
    ruled table 98  
    spice catalog 150  
    wedding invitation 54  
    word wrapping by column 140  
Scripting Examples folder 7  
scripts 49, 51, 157  
serial number 9  
set defaultisstyled statement 172  
set filename command 37, 118  
set filename statement 172  
set firstmaster statement 132  
set fn of pic command 118  
set height of picture statement 123  
set itemdelimiter statement 110, 172  
set master statement 132, 172  
set picturefolders statement 124, 172  
set pictureimportcrop statement 118, 173  
set pictureimportpage statement 118, 173  
set pictureposition statement 173  
set statement 172, 174  
    autostart keyword 174  
    clipboard keyword 174  
    endrecord keyword 174  
    fieldquote keyword 174  
    fieldseparator keyword 174  
    filetype keyword 174  
    fldsep keyword 174

ignoremissingpictures keyword 174  
    options for 172, 174  
recordseparator keyword 174  
startrecord keyword 174  
subfieldseparator keyword 174  
viewfrequency keyword 174  
set width of picture statement 123  
set wordcharacters statement 173  
set worddelimiters statement 173  
set xmargin of picture statement 123  
set xoffset of picture statement 123  
set xscale of picture statement 123  
set ymargin of picture statement 123  
set yoffset of picture statement 123  
set yscale of picture statement 123  
solutions to common problems 185-187  
special characters 161  
spreadsheet applications 83  
story 48, 153  
    searching for 155  
string concatenation operators 107  
styled keyword 143  
styled text 143  
subfield operator 111, 178  
subfield separator 174  
substory 49, 153  
    searching for 155  
substrings 107

## T

tab constant 179  
tab-delimited formal 69  
tab-delimited format 77  
technical information 181  
trim function 112, 140, 178  
true constant 179  
Tutorial folder 7

## U

Unicode 165  
unquoted keyword 144  
unstyled keyword 144  
upcase function 112, 178

**V**

validating data 150  
variables 75, 137  
VBScript 46, 50  
View Preferences panel 165  
Visual Basic 158  
Visual FoxPro 81

**W**

Windows vs. Mac differences 4  
word characters 173  
word delimiters 173  
word operator 108, 178  
wordcase function 112, 178

**X**

XPress Tags 143, 171