



DDF Builder User's Guide

Zen v15

Activate Your Data™



Copyright © 2023 Actian Corporation. All Rights Reserved.

This Documentation is for the end user's informational purposes only and may be subject to change or withdrawal by Actian Corporation ("Actian") at any time. This Documentation is the proprietary information of Actian and is protected by the copyright laws of the United States and international treaties. The software is furnished under a license agreement and may be used or copied only in accordance with the terms of that agreement. No part of this Documentation may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or for any purpose without the express written permission of Actian. To the extent permitted by applicable law, ACTIAN PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, AND ACTIAN DISCLAIMS ALL WARRANTIES AND CONDITIONS, WHETHER EXPRESS OR IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OR OF NON-INFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT WILL ACTIAN BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF ACTIAN IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The manufacturer of this Documentation is Actian Corporation.

For government users, the Documentation is delivered with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013 or applicable successor provisions.

Actian, Actian DataCloud, Actian DataConnect, Actian X, Avalanche, Versant, PSQL, Actian Zen, Actian Director, Actian Vector, DataFlow, Ingres, OpenROAD, and Vectorwise are trademarks or registered trademarks of Actian Corporation and its subsidiaries. All other trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

About This Document	vii
Who Should Read This Manual	vii
Getting Started with DDF Builder	1
DDF Builder Overview	2
Why Use DDF Builder	2
What DDF Builder is Not.	3
Why Not Use ZenCC?	3
What You Need to Know	3
Components of DDF Builder	4
Starting DDF Builder	6
Starting DDF Builder from a Command Line	6
Where to Go From Here.	9
Using DDF Builder	11
DDF Builder Concepts	12
Terminology Review	12
Security.	13
Previous Database and DDF Versions	13
DDF Builder Error Detection and Correction	13
Saving Original and Modified Definitions.	15
Table Definition Editor Pages	15
Legacy Nulls in DDF Builder.	17
True Nulls in DDF Builder.	17
GUI Reference	19
Welcome Page	20
Data Sources Explorer	21
Btrieve File Editor	21
Table Definition Editor.	23
Table Page	24
Indexes Page.	28
Preview Page	29
Statistics Page.	29
SQL View Page.	30
Add Database	31
Check Table Consistency	32

Copy SQL Definition	34
Export Btrieve Schema	34
Import Btrieve Schema	35
Add Data Path	36
Change Associated Data File	36
Btrieve Types	36
Definition Errors	38
Original Definition	40
DDF Builder Tasks	42
General Tasks	42
Tasks Initiated from Data Sources Explorer	43

DDF Builder Tutorials **49**

Using the DDF Builder Tutorials	50
Tutorial 1 Overview	50
Tutorial 2 Overview	50
Before You Begin	50
Tutorial 1 – Creating Table Definitions with DDF Builder	53
Scenario	53
Goals	53
What You Need to Know	53
Create a Zen Database	54
Open an Existing Btrieve File	56
Review DDF Builder’s Findings	58
Define the Record Fields	59
Review the Index Information	68
Preview the Defined Data	69
Conclusion	70
Tutorial 2 – Modifying Table Definitions with DDF Builder	71
Scenario	71
Goals	71
Lesson 1 – Working with v3.00 DDFs	72
Scenario	72
Goals	72
What You Need to Know	72
Open the Btrieve File	72
Understanding the Warning Message	73
How To Convert My Files	73
Conclusion	76
Lesson 2 – Working with Pre v6.x File Formats	77

Scenario	77
Goals.....	77
What You Need to Know	77
Open the Btrieve File	77
Understanding the Warning Message.....	78
View the Log File.....	78
How To Rebuild My Files	79
Where To Go From Here	79
Conclusion	79
Lesson 3 – Invalid Data Types and Sizes	80
Scenario	80
Goals.....	80
What You Need to Know	80
Open the Btrieve File	80
Look for Inconsistencies.....	82
Understanding the Errors	83
Review Data Types and Sizes.....	84
Make the Final Changes.....	88
Save the Table Definition.....	90
Conclusion	90
Lesson 4 – Overlapping Column Definitions.....	91
Scenario	91
Goals.....	91
What You Need to Know	91
Open the Btrieve File	91
Look for Inconsistencies.....	93
Understanding the Errors	94
Accept or Reject Changes	95
Save the Table Definition.....	95
Conclusion	96
Lesson 5 – File/Field Flag Inconsistencies.....	97
Scenario	97
Goals.....	97
What You Need to Know	97
Open the Btrieve File	97
Look for Inconsistencies.....	99
Understanding the Errors	100
Accept or Reject Changes	101
Save the Table Definition.....	102
Conclusion	102

Lesson 6 – Index Inconsistencies	103
Scenario	103
Goals	103
What You Need to Know	103
Open the Btrieve File	103
Look for Inconsistencies	105
Understanding the Errors	106
Name the Index	107
Save the Table Definition	107
Conclusion	108
Lesson 7 – Variable Length Record Mismatch	109
Scenario	109
Goals	109
What You Need to Know	109
Open the Btrieve File	109
Look for Inconsistencies	111
Understanding the Errors	112
Define the Unknown Field	113
Save the Table Definition	113
Conclusion	114
Lesson 8 – Record Length Mismatch	115
Scenario	115
Goals	115
What You Need to Know	115
Open the Btrieve File	115
Look for Inconsistencies	117
Understanding the Errors	118
Define the Field	119
Save the Definition	120
Conclusion	120

About This Document

This documentation is intended as a guide for working with DDF Builder. Topics include an overview of the utility, components that make up the graphical user interface, and tutorials to aide you when working with the various issues DDF Builder is designed to detect.

Who Should Read This Manual

This manual provides information for users who use DDF Builder. The procedures described in this manual are considered advanced and require a certain amount of knowledge about the structure of your data, as well as some basic knowledge about the transactional and relational access methods.

Any specific concepts or methods that you may need to know are clearly stated in this manual, where appropriate. If you do not have an understanding of the suggested information, you should avoid using DDF Builder until the time when you may obtain that knowledge.

We would appreciate your comments and suggestions about this manual. As a documentation user, you are in a unique position to provide ideas that can have a direct impact on future releases of this and other manuals. If you have comments or suggestions, post your request at the community forum on the Actian website.

Getting Started with DDF Builder

The following topics prepare you to use DDF Builder for relational access to Btrieve files:

- [DDF Builder Overview](#)
- [Starting DDF Builder](#)

DDF Builder Overview

DDF Builder is a Java utility that allows you to view, create, and change Zen data dictionary files (DDFs) without making modifications to the underlying data file, called a Btrieve file. Although DDF Builder provides a variety of functions, the utility's primary purpose supports the following:

- Creating the table definitions required to enable relational access to existing Btrieve data files
- Modifying existing table definitions to ensure that relational access is enabled correctly for Btrieve data files

Note: When creating and modifying DDFs with DDF Builder, the Btrieve data file is never modified.

Other functionality in DDF Builder includes, but is not limited to, the following:

- Creating new Btrieve data files to use with the MicroKernel Engine
- Exporting Btrieve schema to an XML file
- Importing Btrieve schema from an XML file
- Creating SQL statements

Why Use DDF Builder

DDF Builder is a specialized utility that allows you to add relational access to your Btrieve data files without making changes to the Btrieve data files. Typically DDF Builder is not a utility that you would use daily. Instead, DDF Builder should be used as needed to add relational access to an existing Btrieve file by creating table definitions in data dictionary files, or by modifying existing data dictionary files to connect table definitions that were not properly constructed.

Note: DDFs define the schema for your SQL metadata. DDFs are system tables that allow DDF Builder to represent SQL access as objects, or SQL tables. Rather than modifying DDFs directly, you work with the SQL table objects. DDF Builder modifies the DDFs as you create, change, or delete SQL tables. All of the SQL tables in a database are defined in the same set of DDFs.

Some previous versions of the Table Designer in the Zen Control Center provided two modes in which to work – *Linked* and *Unlinked*. The linked mode allowed you to make changes to both the table definitions and the data file, and the unlinked mode allowed you to make changes to only the table definitions. DDF Builder uses a similar unlinked mode by using IN DICTIONARY calls to modify the DDFs without making changes to the Btrieve file. DDF Builder never writes to the Btrieve file.

What DDF Builder is Not

DDF Builder is not a means by which you can modify your existing Btrieve data files. If you intend to modify existing Btrieve files *and* DDFs that provide relational access, DDF Builder is not the utility you need to use. Consider using Zen Control Center and other Zen utilities for changing your existing Btrieve files.

Additionally, DDF Builder is not intended to be used to create or modify keys. Use the Table Editor in Zen Control Center when working with keys. DDF Builder is intended to create and modify DDFs.

Why Not Use ZenCC?

Zen Control Center is intended to manipulate the physical data files and the data dictionary files at the same time, or in linked mode. Using ZenCC to alter only DDFs is not recommended.

What You Need to Know

A thorough understanding of the structure of your data is the most important thing you need to know when using DDF Builder. Without an understanding of how the data is structured, creating or modifying table definitions with DDF Builder could be difficult, time consuming and counterproductive.

Other Helpful Information

It is also helpful if you have a general understanding of the two primary methods in which data is accessed from Zen databases – transactional and relational. For much of the functionality in DDF Builder, an understanding of transactional access is beneficial. Other areas of functionality require that you are familiar with relational database concepts in general.

Note: This book assumes knowledge of the transactional access method and relational concepts. Listed below are Zen books to add to your understanding before attempting to use DDF Builder. Also, Zen developer documentation is installed with the Zen database engine.

Transactional Access

With transactional access, an application program navigates up and down along either physical or logical pathways through data records. Using a transactional API, an application program provides direct control and allows a developer to optimize data access based on knowledge of the underlying structure of the data. Using the Btrieve API is an example of transactional access.

Refer to the following books to learn more about transactional access:

- *Zen Programmer's Guide* (Developer Reference)
- *Btrieve API Guide* (Developer Reference)
- *Advanced Operations Guide* (Advanced Reference)

Relational Access

Relational is an access method in which data is represented as collections of tables, rows, and columns. The relational model insulates the developer from the underlying data structure and presents the data in a simple table format. ODBC is an example of a relational access method.

Refer to the following books to further your understanding of relational access:

- *SQL Engine Reference* (Advanced Reference)
- *Advanced Operations Guide* (Advanced Reference)
- *Zen User's Guide* (General Reference)
- *Zen Programmer's Guide* (Developer Reference)

What To Do Next

Changes made to your table definitions with DDF Builder will alter the structure of your DDFs. As a precaution before using the utility, always back up any of the files (including DDFs) with which you intend to work. (Btrieve files are also referred to as data files because the data is stored within the page structure of the file.)

Note: DDF Builder does not allow you to modify the record layout structure of *existing* Btrieve files. You can create *new* Btrieve files with the utility, if you choose.

Disable Security

If you are working with a database that has any of the Zen security models enabled, you should take the database offline and disable all security before opening the files in DDF Builder.

Components of DDF Builder

In addition to the utility, DDF Builder includes the following components:

Log File

The .log file is where DDF Builder enters reports of any problematic conditions. In a default installation, the log file is installed in the following directory:

C:\ProgramData\Actian\Zen\rcp\workspace-builder\.metadata

User Documentation

DDF Builder User's Guide is part of the Zen user documentation.

Tutorial Files

DDF Builder installs a set of files for use with the tutorials included in *DDF Builder User's Guide*. The tutorial files are installed by default in

C:\ProgramData\Actian\Zen\DDFBuilder\tutorials. For documentation, see [DDF Builder Tutorials](#).

For a list of default installation locations, see [Where are the files installed?](#) in *Getting Started with Zen*.

Starting DDF Builder

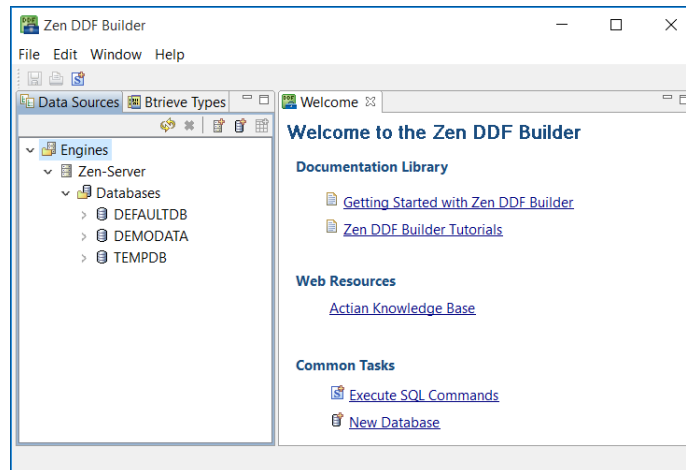
You can start DDF Builder from the Windows operating system, from within the Zen Control Center (ZenCC), or from a command line.

To start DDF Builder from the Windows Operating System

1. Access **DDF Builder** from the **Start** menu or **Apps** screen.
The **DDF Builder** main window opens.

To start DDF Builder from ZenCC

1. In ZenCC, click **Tools > DDF Builder** from the menu bar.
2. The **DDF Builder** main window opens.



Starting DDF Builder from a Command Line

DDF Builder can be started on Windows, Linux, or macOS by running the executable file from a command line. See the following topics for details.

Windows

Run the executable file **builder.exe** to start DDF Builder. See [Where are the files installed?](#) in *Getting Started with Zen*.

Linux and macOS

Run the **executable script file builder to start** DDF Builder. The **script** file is located, by default installation, in the `usr/local/actianzen/bin` directory.

We recommend that you start DDF Builder from a command prompt and **not** by double-clicking the script file using a file browser application. See [Troubleshooting Guide for Running DDF Builder](#).

Requirements for Starting DDF Builder on Linux or macOS

The following requirements must be met to start DDF Builder on Linux or macOS:

Requirement	Discussion
Zen server or client	A compatible Zen server or client must already be installed on the same machine. See Installing Zen for Linux-based Systems in <i>Getting Started with Zen</i> .
X server access	The <code>xhost</code> command controls which clients can access X Window System on the current machine. By default, <code>xhost</code> turns on access control. This means that only the user who starts X Window System could start DDF Builder. You may turn off X Window System client restrictions by running <code>xhost +</code> in a terminal window.
Java Runtime Environment (JRE)	The JRE components required to run DDF Builder are installed as part of Zen. DDF Builder uses the “local” version of the JRE installed as part of Zen.
Desktop ownership (macOS only)	Only the user logged in to the Desktop can start DDF Builder.

Troubleshooting Guide for Running DDF Builder

If you have met the requirements to run DDF Builder and still are having difficulty running the utility, refer to the following guidelines:

Troubleshooting Condition	Discussion
You receive the error "java.lang.UnsatisfiedLinkError."	<p>This error typically occurs if you try to start DDF Builder by double-clicking the script file using a file browser application. Start DDF Builder from a command prompt.</p> <p>This error can result if the LD_LIBRARY_PATH variable is not set. The builder script sets this variable for you. You may also explicitly set the variable with the following command:</p> <pre>export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:/usr/local/actianzen/lib64</pre> <p>On macOS, the variable is DYLD_LIBRARY_PATH.</p>
You receive the error "SWT no more handles" when trying to run DDF Builder as root or as user zen-svc.	<p>You are not required to log in as user zen-svc or root to run DDF Builder. However, if you are neither of these users, you must be a member of group zen-data.</p> <p>The "SWT no more handles" error is caused by the X server denying a connection to a client. Before switching to user zen-svc or root, open a console window and type <code>xhost +</code> to allow other clients to connect to the X server.</p> <p>Now you can switch to user zen-svc or root.</p> <p>Also, sometimes the display environment variables needs to be set. As user zen-svc or root, type the following command at a console window:</p> <pre>export DISPLAY=:0.0</pre> <p>or</p> <pre>export DISPLAY=localhost:0.0</pre>
You want to view the error log file for DDF Builder or redirect the errors to the console window.	<p>The log file of DDF Builder errors is located under the user's home directory. For troubleshooting, you may find it more convenient to redirect the errors to the console window.</p> <p>To redirect errors to the console window, use the <code>-consoleLog</code> option when starting DDF Builder:</p> <pre>builder -consoleLog</pre>
You receive the following error message: "Unable to connect to database engine. Make sure the target machine is accessible and an engine is running on the target machine."	<p>The context of this error occurs if you attempt to administer the local server.</p> <p>To administer the local server, you must be a member of the zen-data group or be the root user. See Zen Account Management on Linux, macOS, and Raspbian in <i>Getting Started with Zen</i>.</p>

Troubleshooting Condition	Discussion
You receive the error: “GTK IM Module SCIM: Cannot connect to Panel!” then trying to run DDF Builder as a user other than root.	<p>On some Linux operating systems, it is necessary to specify the environment variable <code>GTK_IM_MODULE</code>.</p> <p>To resolve this problem, before starting DDF Builder, run the following command at the console window:</p> <pre>export GTK_IM_MODULE=scim-bridge</pre>

Clearing the DDF Builder Cache

DDF Builder caches certain information to improve efficiency. On occasion, you may need to clear the cache for troubleshooting purposes as directed by technical support or to ensure that all files are reloaded. The cache can be cleared only by starting DDF Builder with a parameter from the command line.

Note: Clearing the cache when starting DDF Builder provides no advantage under normal usage. For normal usage, start the utility as discussed in [Starting DDF Builder](#).

To clear DDF Builder cache

1. Exit DDF Builder if it is running.
2. Open a command prompt.
3. Change directory to the Zen\bin\ folder in the Zen installation directory.
4. Enter `builder -clean` and press **Enter**.

This clears the cache and starts DDF Builder.

Where to Go From Here

Now that you have a general understanding of DDF Builder, see [Using DDF Builder](#) for an introduction to the user interface.



Using DDF Builder

The following topics cover concepts of use, the user interface, and things you can do with DDF Builder:

- [DDF Builder Concepts](#)
- [GUI Reference](#)
- [DDF Builder Tasks](#)

DDF Builder Concepts

This section contains conceptual information about working with DDF Builder. The following sections are included:

- [Terminology Review](#)
- [Security](#)
- [Previous Database and DDF Versions](#)
- [DDF Builder Error Detection and Correction](#)
- [Saving Original and Modified Definitions](#)
- [Table Definition Editor Pages](#)
- [Legacy Nulls in DDF Builder](#)
- [True Nulls in DDF Builder](#)

Terminology Review

Before you begin working with DDF Builder, you should review the terminology used throughout the manual. Some of the terms used are specific to the relational database model, whereas some of the information is from the transactional database methodology. Because DDF Builder is designed to create relational table definitions from transactional data, the utility brings together these two models in one place. This information is useful because terminology used in the transactional model may be referred to differently in the relational model. The following table lists the terms used throughout this manual.

Terms	Definition
Database	A collection of files that includes a set of DDFs with table definitions, a set of data files and a DSN.
File Table	The Btrieve physical file or SQL table name.
Dictionary Data Dictionary System Catalog System Tables	A set of DDF files that can contain table definitions of the data (metadata).
Table Definition	Entry in the DDFs that corresponds to a Btrieve file.

Terms	Definition
DDFs System Tables System Objects	Files with a .DDF extension. DDFs are system tables that provide the means for defining files within the constraints of a relational database (the metadata).
Key Index	Specified data used to sort a table or file in a particular order and to optimize searches on particular values.

Security

When using DDF Builder with security-enabled databases, it is recommended that you turn off all security prior to opening the files in DDF Builder. Failure to turn off security results in DDF Builder sending secure login and password requests each time a call is executed. This could become excessive and counterproductive, depending on the number of calls needed to create or modify the table definition.

Previous Database and DDF Versions

DDF Builder is supported on PSQL v9 and later versions. DDFs created with Scalable SQL v4.00 and later are supported along with Btrieve data files v6.x and later.

Scalable SQL v3.xx and Earlier

If you have DDFs that were created with Scalable SQL v3.xx or earlier, you must convert your DDFs before you can open them with DDF Builder.

For step-by-step instructions, see [Lesson 1 – Working with v3.00 DDFs](#).

Btrieve v5.x and Earlier

DDF Builder supports version 6.x files but does not support files created prior to version 6.x. If you have Btrieve data files that were created with Btrieve v5.x or earlier, you must rebuild the files before you can open them with DDF Builder.

For step-by-step instructions, see [Lesson 2 – Working with Pre v6.x File Formats](#).

DDF Builder Error Detection and Correction

When you open an existing table definition associated with a Btrieve file, DDF Builder goes through a set of comparisons and checks. From these comparisons and checks, DDF Builder may

make certain adjustments to the table definition, if errors are detected. Any changes made to the table definition are recorded in the Definition Errors window.

Note: DDF Builder always retains the original table definition, so the changes made by DDF Builder are never automatically saved to your table definition. You may save either the modified or the original table definition with a different name to retain both definitions.

So that you can understand the assessments and potential changes DDF Builder makes, take a moment to look at the steps DDF Builder goes through.

First, DDF Builder reviews the Btrieve file and examines the pertinent details, such as record length, indexes, index segments, and so forth. Next, DDF Builder opens the existing DDFs and looks at the information contained in the DDFs in comparison to the Btrieve file. These two sets of details are then systematically compared to each other for matches and mismatches.

DDF Builder uses the details found in the Btrieve file as the basis for detecting and fixing any errors or problems. The following examples help clarify.

Example 1 - Index Inconsistencies

You have a Btrieve file that contains a key. The table definition does not contain a corresponding index. Since DDF Builder cannot alter your Btrieve file, the key defined in the Btrieve file is carried over to your table definition as an index.

For information on dealing with this type of situation, refer to [Lesson 6 – Index Inconsistencies](#).

Example 2 - Record Length Mismatch

Your Btrieve file contains records defined with a total length of 120 bytes, yet your table definition only defines a total of 100 bytes. Since DDF Builder cannot alter the Btrieve record, a column is added to the table definition for the 20 bytes unaccounted for in the definition.

For information on dealing with this type of situation, refer to [Lesson 8 – Record Length Mismatch](#).

Example 3 - Flag Inconsistencies

The Btrieve file contains a flag on a key that does not match the flags set on the corresponding SQL index in the table definition. Again, because DDF Builder cannot change the Btrieve file, the flag setting on the SQL index in the table definition is changed to reflect the setting in the Btrieve file.

For information on dealing with this type of situation, refer to [Lesson 5 – File/Field Flag Inconsistencies](#).

Note: The topic [DDF Builder Tutorials](#) provides step-by-step instructions on dealing with the various situations you may encounter when working with DDF Builder.

DDF Builder cannot repair every problem. For a complete list of the issues that DDF Builder detects and repairs, see [Definition Errors List](#).

Saving Original and Modified Definitions

Although DDF Builder makes recommended changes to your existing table definitions, you are not required to keep the changes, nor are you required to discard your original table definition if you accept the changes.

You can save both your modified and original table definitions by saving one of the definitions with a different name. If you accept the changes made by DDF Builder, you may save either the modified or the original table definition with a different name.

If you reject the changes made by DDF Builder, your original table definition is retained with the original name.

Table Definition Editor Pages

The Table Definition Editor is made up of several pages of information that you use for viewing, creating, and modifying table definitions. The Table Definition Editor contains the following pages:

- Table Page
- Indexes Page
- Preview Page
- Statistics Page
- SQL View Page

Table Page

The Table page is where you do most of the work when creating or modifying table definitions. The Table page includes the raw data view and the grid data view.

Raw Data View

The raw data view displays data in your Btrieve file in a combined ASCII and hexadecimal view. Record length, offsets, and field sizes are also shown in this view.

The raw data view provides you with the visual indicators you may need to determine columns and null indicators, as well as identify unknown fields and bytes.

Tip... For information on the visual indicators used in the raw data view, refer to [The following table lists the attributes displayed in the raw data view.](#)

Grid Data View

The grid data view is similar in feature and functionality to the grid window view in the SQL Editor in Zen Control Center.

Fields used in indexes cannot have their size or type modified; only the field name can be modified in the grid data view. To add indexes, drop indexes, or change other index information, you would need to modify the Btrieve file directly using the Btrieve Maintenance Utility.

Tip... For information on the visual indicators used in the grid data view, refer to [The following table lists the attributes displayed in the grid data view.](#)

Indexes Page

The Indexes page allows you to see the indexes and index segments that DDF Builder detected in the Btrieve file. You may not add or modify any indexes or index segments from this page. Only index name changes are allowed on the Indexes page. If you change the name of an indexed field on the Table page, the name is updated in real-time on the Indexes page.

Note: For the steps needed to name an index, see [Name an Index](#).

Alternate Collating Sequence (ACS) Files

When working with Btrieve files that use alternate collating sequence (ACS) files, the ACS file must reside in the same directory as the Btrieve file and must have an .alt extension, as in upper.alt.

Preview Page

The Preview page provides a look at file data formatted using the current table definition. As you make changes to the table definition, the preview changes accordingly.

The Preview page is read-only. You may not edit any of the information displayed on this page. Although this page is read-only, you may move through the records by using the arrow buttons located at the bottom of the page.

Statistics Page

The Statistics page displays the Btrieve file statistics for any file open in the Table Definition Editor. The information presented on this page would be the same information reported if a statistics report were generated on the file using the Btrieve Maintenance utility.

SQL View Page

The SQL View page displays the underlying SQL statements that apply to the creation or modification of your table definitions. As you modify your table definition, the underlying SQL statements are updated immediately on the SQL View page.

The information on this page is read-only and cannot be altered on this page. You may select and copy the statements to use elsewhere, if needed.

Caution! Do not reuse any statement that contains a reference to any of the Zen dictionary system objects. These objects are easily identified by their X\$<tablename> and include a proprietary comment prohibiting reuse.

Legacy Nulls in DDF Builder

Versions of Zen prior to Pervasive.SQL 2000 only provided support for legacy nulls. Files that use legacy nulls are recognized by DDF Builder, but no action is required to deal with this type of null when working with DDF Builder.

True Nulls in DDF Builder

In Pervasive.SQL 2000, support was added for true nulls. With true nulls, a leading byte, known as a null indicator byte, is included to designate the corresponding column as having a null or non-null value

Working with True Nulls

When defining record fields within DDF Builder, it is important to know the fields in the record that are nullable, since marking a section of the record as nullable adds an extra byte to the field. This is because in a Btrieve file, nullable portions of the record are designated as such by the use

of the null indicator byte. In the raw data view of the Table Definition Editor, the null indicator byte is indicated in the byte immediately preceding the field or column. When the Null check box is selected, the null indicator byte becomes active and the size of the field is automatically reduced by one byte to accommodate for that byte.

Note: If you are working with files created prior to Pervasive.SQL 2000, nulls are most likely not an issue.

Creating Nullable Fields

Creating nullable fields in DDF Builder requires that you consider the size of the field and the one byte needed for the null indicator. This means that a nullable field intended to be 25 bytes would actually be defined as 26, reserving one byte for the null indicator. Once you designate a field as nullable, the size is automatically reduced in DDF Builder by one.

Note: For examples of working with nullable and non-nullable columns, see the examples in Tutorial 1, [Create a Nullable Column in the raw data view](#) and [Change a Non-Nullable Column to Nullable in the grid data view](#).

GUI Reference

DDF Builder provides a graphical user interface (GUI). The GUI includes various editors, views, and wizards with which you display and work with objects.

The object being edited is represented by a tab on top of the editor. The tab contains the name of the object. Data modified within an editor must be explicitly saved.

Views can be opened only one at a time. Actions performed within a view are applied immediately. No explicit save is required.

Wizards contain one or more dialogs and guide you through a task to obtain a specific result.

The following table lists the editors, views, and wizards that DDF Builder provides.

GUI Component	Editor	View	Wizard	Description
Data Sources Explorer		X		Data Sources Explorer
Btrieve File Editor	X			Btrieve File Editor
Table Definition Editor	X			Table Definition Editor
SQL Editor	X			SQL Editor in <i>Zen User's Guide</i> ¹
Grid Window View		X		Grid Window View in <i>Zen User's Guide</i> ¹
Text Window View		X		Text Window View in <i>Zen User's Guide</i> ¹
Outline		X		Outline Window View in <i>Zen User's Guide</i> ¹
Add Database			X	Add Database
Check table consistency (Check Database and Check Tables)		X ²		Check Table Consistency
Copy SQL Definition			X	Copy SQL Definition
Export Btrieve Schema			X	Export Btrieve Schema
Import Btrieve Schema			X	Import Btrieve Schema
Change Associated Data File			X	Change Associated Data File
Btrieve Types		X		Btrieve Types
Definition Errors		X		Definition Errors

GUI Component	Editor	View	Wizard	Description
Original Definition		X		Original Definition

¹DDF Builder and ZenCC share common components. Because of this, some editors, views, and wizards are documented in other Zen books and not in *DDF Builder User's Guide*.

²The Check Database and Check Table action is similar to a wizard except that you make your selection in Data Sources Explorer rather than in a dialog. The results generated by checking consistency are similar to a view.

Welcome Page

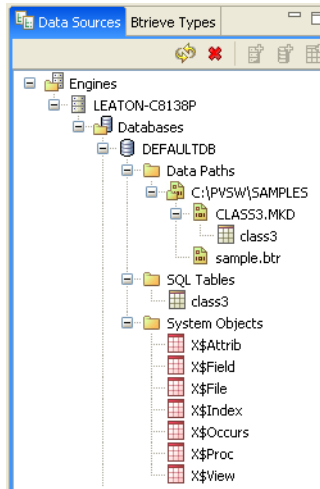
When you start DDF Builder, it displays the Welcome tab, where you can access a variety of information and perform common tasks, such as adding a new server and creating a new database. The information provided includes the following:

- Getting Started with Zen DDF Builder
- Zen DDF Builder Tutorials
- Actian Knowledge Base
- Common Tasks

If Welcome tab is closed, you can open it in the DDF Builder menu by selecting **Help > Welcome**.

Data Sources Explorer

DDF Builder uses a file explorer, a tree of objects called the Data Sources Explorer. The objects include databases, data paths, Btrieve files, and SQL tables. The objects are referred to as *nodes* in the tree.



Nodes can be expanded or collapsed to reveal or conceal subordinate nodes. The expand/collapse icon appears to the left of the node if a lower node is available.

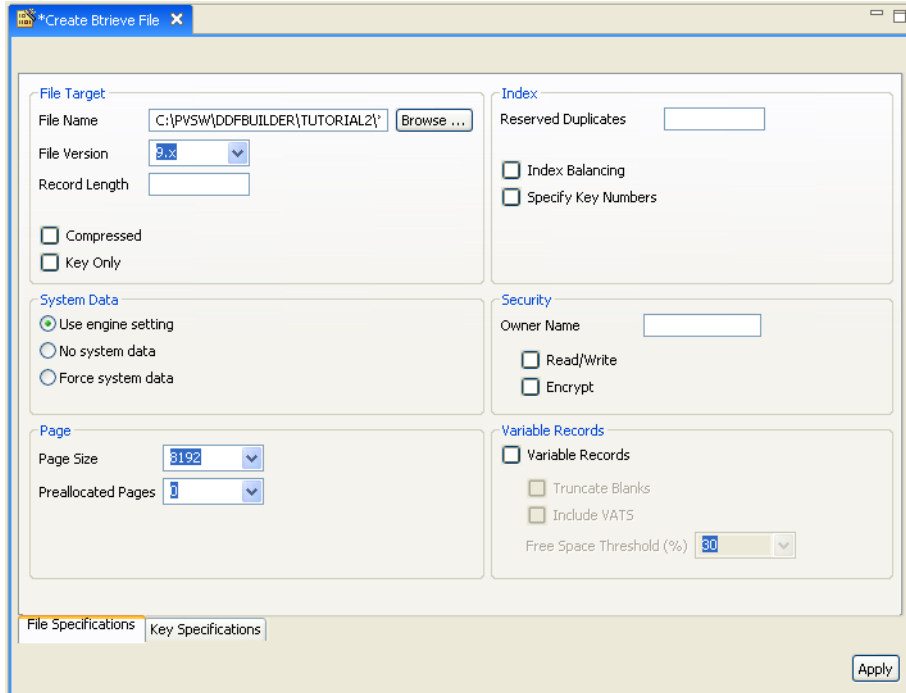
Btrieve File Editor

Btrieve File Editor creates the file and key specification for a new Btrieve file. DDFs are not automatically created for a new Btrieve file. You must add them separately, if you want them (see [SQL Tables](#)).

Tip... To create both a Btrieve file and its DDFs, use SQL Editor in Zen Control Center.

How to Access

In Data Sources Explorer, right-click a data path under the **Data Paths** node and select **Create Btrieve File** to open the Btrieve File Editor.



A Btrieve file must be located on physical storage, which is why the editor is invoked from a data path.

Features

Btrieve File Editor contains two tabs: one for specifying the characteristics of the file and one for specifying characteristics of the keys. The **Apply** button on each tab saves the specifications for that tab. Click **File > Save** to save the editing changes for both tabs.

At a minimum, you must supply a file name and a record length to create a Btrieve file. You specify these on the File Specifications tab.

As stated in [What You Need to Know](#), this book assumes that you thoroughly understand the transactional access method and relational concepts. This section does not attempt to explain the controls on the Btrieve File Editor tabs. The following table provides links to related documentation if you need to further your understanding of Btrieve files.

Tab	Advanced Reference Documentation	Developer Reference Documentation
File Specifications	In <i>Advanced Operations Guide</i> : <ul style="list-style-type: none"> • File Size • Owner Names and Security • System Data • File Information Editor 	In <i>Zen Programmer's Guide</i> : <ul style="list-style-type: none"> • Btrieve Fundamentals • Designing the Database
Key Specifications	In <i>Advanced Operations Guide</i> : <ul style="list-style-type: none"> • Methods for Handling Duplicate Keys 	In <i>Zen Programmer's Guide</i> : <ul style="list-style-type: none"> • Btrieve Fundamentals • Designing the Database • Working with Records • Creating a Database

Restrictions

Only one Btrieve file can be created at a time. On the Key Specifications tab, click **Apply** after adding or changing each segment of a segmented key. This saves the changes to each segment before you create or edit the next segment.

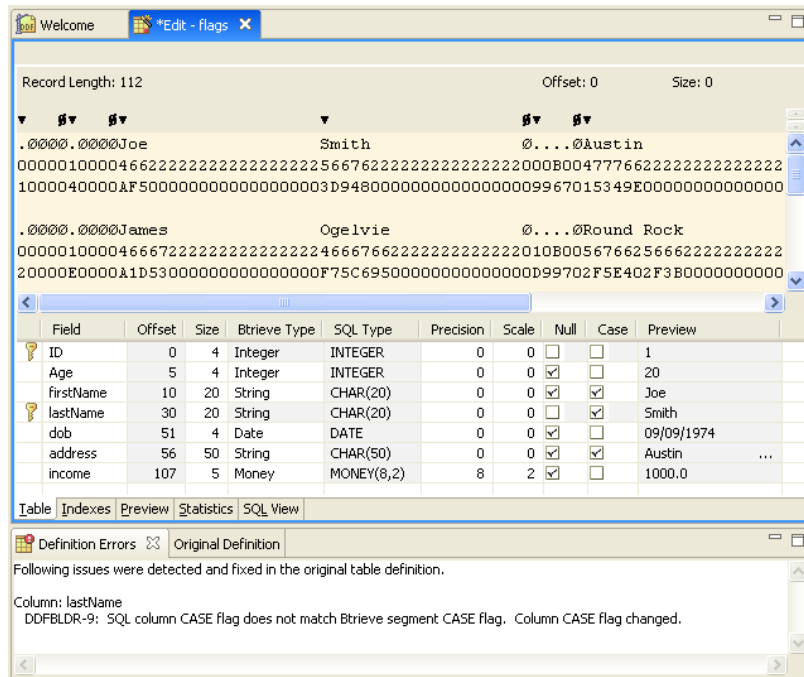
Table Definition Editor

The Table Definition Editor creates new or changes existing schemas for SQL metadata. Note that Table Definition Editor resembles Table Editor in Zen Control Center (ZenCC) and is used in similar ways. For more information, see [Table Editor](#).

How to Access

Do one of the following in Data Sources Explorer:

- Right-click a Btrieve file name and select **Create Table Definition**.
- Double-click a SQL table name to edit an existing table definition.
- Right-click a SQL table name and select **Edit Table Definition**.



Features

The Table Definition Editor, like the Table Editor in Zen Control Center (ZenCC), uses tabs or pages to display different views of table definition information. Some of the pages contain read-only information, while others provide the work area in which you will create and modify your table definitions.

Restrictions

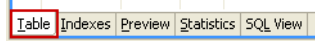
You cannot change the size, offset, or data type for a column if the column is used in a key in the Btrieve file. DDF Builder does not change nor permit changing the layout structure of an existing Btrieve file.

Table Page

The Table page in Table Definition Editor provides two metadata views – raw and grid. The raw view displays information in the Btrieve file, while the grid view displays it as a SQL table.

How to Access

Select the Table page by clicking the **Table** tab at the bottom of the Table Definition Editor.

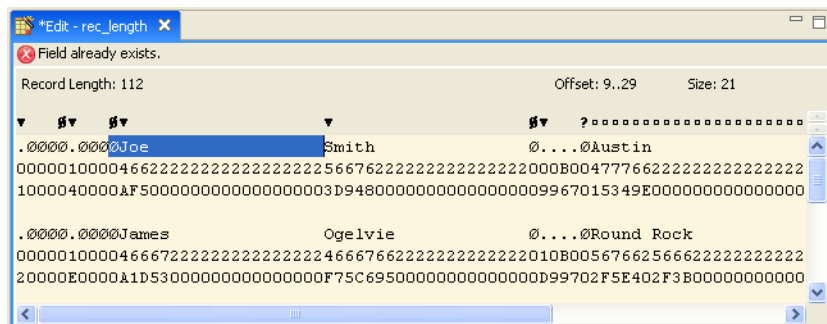


Raw Data View

The raw data view shows both the hexadecimal and ASCII values of data records. This view also displays the record length and the offset and size for selected bytes. Offset and size are adjusted as bytes are selected in this view.





Raw Data View in Table Definition Editor

Field, column, null and unknown indicators appear at the top of the raw data view. Field indicators show where each field within the record begins. Null indicators show where the null indicator byte is located. Unknown fields and bytes are also displayed in the raw data view.



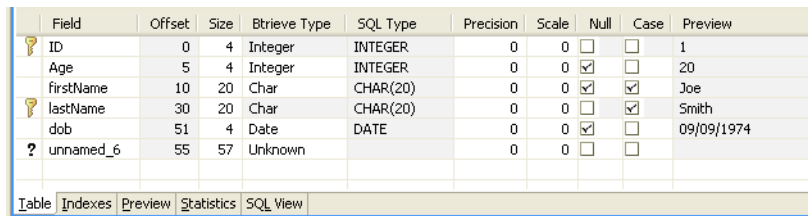
The following table lists the attributes displayed in the raw data view.



Attribute	Description
Error Message	Displays any error or warning condition
Record Length	Displays the entire length of the record, excluding the variable portion.
Offset	Displays the beginning and ending position of the selected bytes or field.
Size	Displays the size (in bytes) of the selected bytes or field.

Attribute	Description
Data Indicators	<p>Displays the field/column indicator () to show where each field within the record begins.</p> <p>Displays the null indicator () if the field is designated as nullable. This byte represents the null indicator byte.</p> <p>Displays the unknown field indicator () if the field has not yet been defined.</p> <p>Displays the unknown byte indicator () if the bytes have not yet been defined.</p>




Grid Data View

The grid data view shows the schema structure of a table as a grid of rows and columns. Each field is represented in a row on the grid. Each row consists of cells that show the attributes for each field. Most of the attributes are editable and can be saved as changes to the schema.



Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
 ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
 firstName	10	20	Char	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	Char	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
? unnamed_6	55	57	Unknown		0	0	<input type="checkbox"/>	<input type="checkbox"/>	

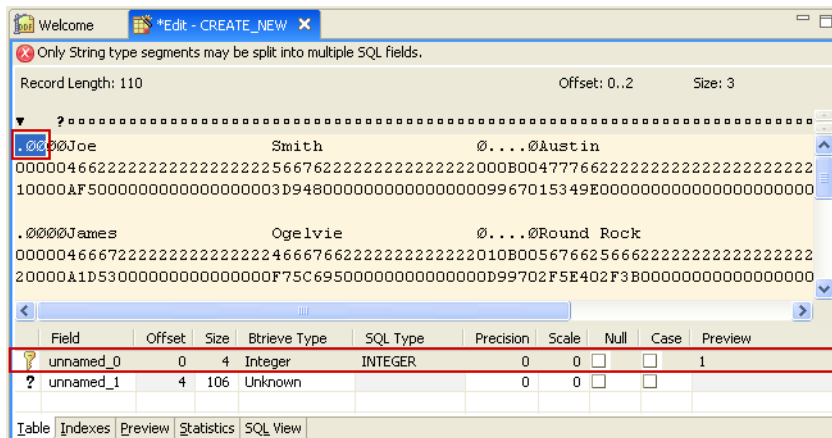
The following table lists the attributes displayed in the grid data view.

Attribute	Description
Column Indicator	<p>Displays the key icon () if the field is used in a key (index) definition.</p> <p>Displays the unknown icon () if the field is unknown.</p> <p>Displays the variable icon () if the field is an unknown variable-length portion.</p>
Field	Displays the table field name.
Offset	Displays the offset, or cumulative, position of the field within the record, starting at 0. View Only
Size	Displays the size (in bytes) of the field.
Btrieve Type	Displays the Btrieve data type for the field.
SQL Type	Displays the SQL data type for the field. View Only
Precision	Displays the number of significant digits for floating point values.

Attribute	Description
Scale	Displays the number of significant digits that are to the right of the decimal point for floating point values.
Null	Selected if the field uses true Null values.
Case	Selected if the field is case insensitive.
Preview	Displays the field contents formatted with the applied data type. View Only

How the Two Views Work Together

Selecting a cell in the grid data view displays the corresponding bytes for that field in the raw data view, just as selecting individual bytes in the raw data view, selects the corresponding elements in the grid data view.



If you try to select past the field definition, an error message appears, and both rows in the grid data view are selected. Both error and warning messages appear in this are of the raw data view.

Table Definition Editor Error Message

Cannot create a field that overlaps another column. Merge the columns and then recreate the fields.

Table Definition Editor Caution/Warning Message

Caution when editing the variable portion. Column Note is a Clob.

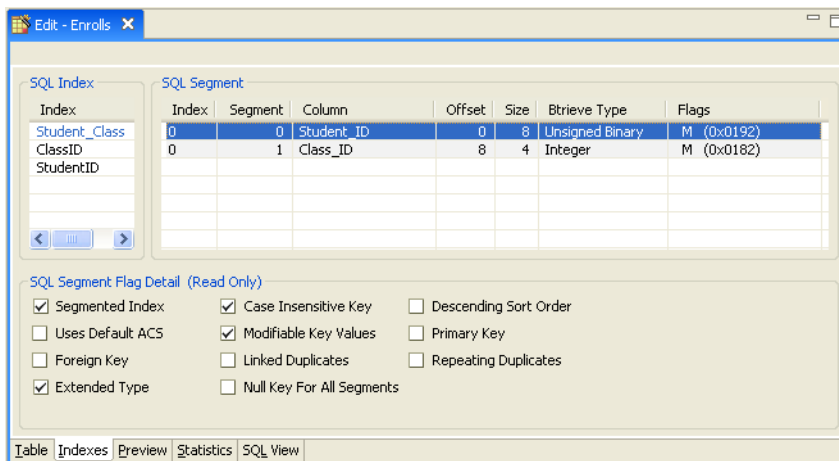
Indexes Page

The Indexes tab is read-only, with the exception of changing the SQL index name. You cannot change the structure of any of the indexes on this tab. Index additions or changes to a SQL table must be made with Zen Control Center. See [Table Editor](#) in *Zen User's Guide*.

How to Access

Select the Indexes page by clicking the **Indexes** tab at the bottom of the Table Definition Editor.

Indexes Page in Table Definition Editor



Features

Name changes made on the Tables page to columns specified as indexes are updated and shown immediately on the Indexes page.

Restrictions

Only the SQL index name may be edited from the Indexes page.

When working with Btrieve files that use alternate collating sequence (ACS) files, the ACS file must reside in the same directory as the Btrieve file and must have an .alt extension (for example, upper.alt).

Preview Page

The Preview page shows file data in a readable layout of columns and rows.

How to Access

Select the Preview page by clicking the **Preview** tab at the bottom of the Table Definition Editor.

Preview Page in Table Definition Editor

Student_ID	Transaction_Num...	Log	Amount_Owed	Amount_Paid	Registrar_ID	Comments
122911800	1	1996-03-28 17:39:33.0	1875.0	0.0	313053054	
123002945	1	1996-03-28 17:39:33.0	1500.0	150.0	313053054	
123233271	1	1996-03-28 17:39:33.0	2000.0	500.0	729772191	
123453193	1	1996-03-28 17:39:33.0	1500.0	150.0	313053054	
123528614	1	1996-03-28 17:39:33.0	4250.0	4250.0	130312616	
123686445	1	1996-03-28 17:39:33.0	1500.0	150.0	313053054	
123771790	1	1996-03-28 17:39:33.0	1600.0	0.0	729772191	
123992115	1	1996-03-28 17:39:33.0	1500.0	150.0	313053054	
124222440	1	1996-03-28 17:39:34.0	3750.0	0.0	313053054	
124442765	1	1996-03-28 17:39:34.0	5100.0	5100.0	130312616	
124759596	1	1996-03-28 17:39:34.0	4500.0	4500.0	313053054	
124981284	1	1996-03-28 17:39:34.0	4000.0	3000.0	729772191	
125201609	1	1996-03-28 17:39:34.0	4750.0	3562.5	320591822	
125431935	1	1996-03-28 17:39:34.0	1500.0	150.0	313053054	
125523080	1	1996-03-28 17:39:34.0	4250.0	4250.0	130312616	

Features

You navigate among data records with the buttons at the bottom center of the page. The file position appears to the right of these buttons. The file position shows how many records are being displayed out of the total number of records. For example, “100-199/1314” indicates that the page displays records 100 through 199 out of a total of 1,314 records.

Changes made on the Tables page to any of the column definitions are reflected immediately on the Preview page.

Information on this page is read-only and cannot be modified.

Statistics Page

The Statistics page displays the file and key specifications for a Btrieve file. The information is read-only; you cannot change it in the view.

How to Access

Select the Statistics page by clicking the **Statistics** tab at the bottom of the Table Definition Editor.

Statistics Page in Table Definition Editor

Statistics									
File Version	9.5	Total Number of Records	211	Available Linked Duplicate Keys	0				
Page Size	4096	Record Length	66	Balanced Key	No				
Page Preallocation	No	Record Compression	No	Total Number of Keys	4				
Key Only	No	Page Compression	No	Total Number of Segments	10				
Extended	No	Variable Records	No						

Keys									
Number	Segment	Offset	Size	Btrieve Type	Flags	Null Values*	Unique Values	ACS	
0	1	0	4	Auto Increment	M (0x0102)	--	211	--	
1	1	4	7	Char	IM (0x0512)	--	211	--	
-	2	11	3	Char	IM (0x0502)	--	211	--	
2	1	58	8	Unsigned Binary	M (0x0116)	--	211	--	
-	2	17	4	Date	M (0x0112)	--	211	--	
-	3	21	4	Time	M (0x0102)	--	211	--	
3	1	29	25	Char	IRMD (0x0593)	--	211	--	
-	2	54	4	Unsigned Binary	RMD (0x0197)	--	211	--	
-	3	17	4	Date	RMD (0x0193)	--	211	--	
-	4	21	4	Time	RMD (0x0183)	--	211	--	

Features

Statistics provides a convenient way to look at the structural characteristics of a Btrieve file. This is particularly useful if you are considering exporting the file schema but are unfamiliar with the file and key specifications.

See also [Export Btrieve Schema](#).

Note: The statistics information is based solely on information in the physical Btrieve file; it does not show any metadata information.

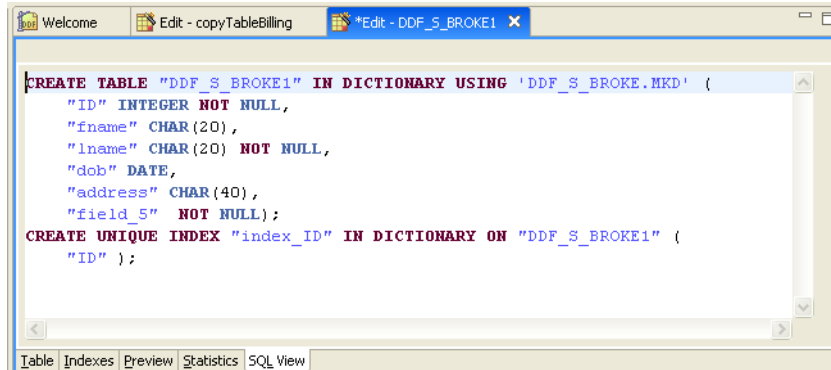
SQL View Page

The SQL View page displays the SQL statement needed to create the current table definition.

How to Access

Select the SQL View page by clicking the **SQL View** tab at the bottom of the Table Definition Editor.

SQL View Page in Table Definition Editor



Features

Most SQL statements created here can be copied and reused in the Zen Control Center for creating new schemas based on the ones you created with DDF Builder. SQL statements that you use in the Zen Control Center can also be saved for future use.

Restrictions

Although the SQL statements created here can be copied and reused, they are not saved, nor is a means for automatically saving them currently offered in DDF Builder. To save the statement, you must copy from this window and save the statement in a text editor.

Caution! Do not reuse any statement that contains a reference to any of the Zen dictionary system objects. These objects are easily identified by their X\$<tablename> and include a proprietary comment prohibiting reuse.

Add Database

The Add Database Wizard creates a new database. This wizard is shared with Zen Control Center (ZenCC). See [Databases](#) and [New Database GUI Reference](#) in *Zen User's Guide*.

Check Table Consistency

DDF Builder provides the ability to check the consistency of a table. A consistency check uses a set of validation rules to compare a physical data file against the metadata in its data dictionary files.

Consistency check validates conditions such as the following:

- Dictionary files have compatible version information.
- Tables have a valid name, ID, and an existing, accessible physical file.
- Columns have valid names, correct total record length, and no overlapping definitions.
- Indexes are correct in name, number, size, data type, offset, and attributes.




You can check the consistency of all tables at once or of tables selected individually. The check reports a count of the validation messages, errors, and warnings by object. An object is the database, a table, or a data dictionary file (DDF).

Validation messages grouped under the heading Passed list the consistency checks that were validated. Errors are grouped under Error and warnings under Warning.

Error messages are always displayed. If an object has both errors and warnings, the warnings are also listed under Error.

You can display or hide the validation messages and the warnings.

Icons on the results view identify the different types of messages:

-  **Validation check**
-  **Error**
-  **Warning**

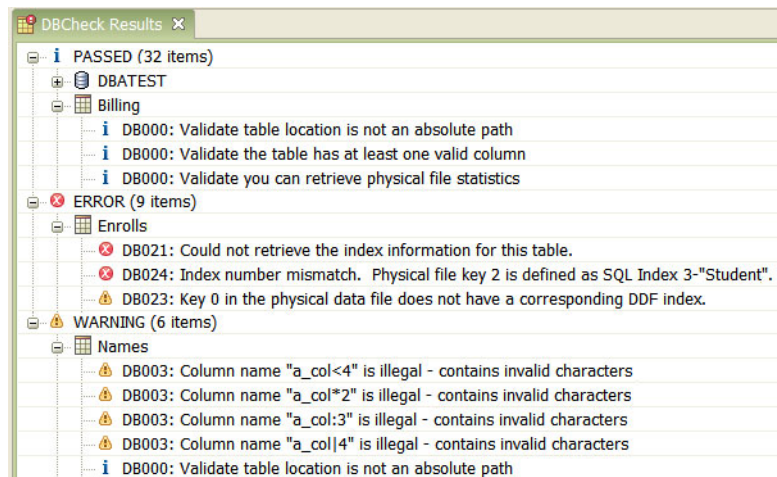
An error indicates a problem in the table definition that will, in most cases, cause a failure or incorrect data to be returned when the data file is accessed. For example, an index defined in the DDFs but not in the data file results in an error. A SQL query that causes the engine to optimize on that particular index generates a failure because no such index actually exists in the data file.

Warnings are indicative of possible problems, but the problem may not cause any failures. For example, an index defined in a data file without a corresponding DDF entry results in a warning. SQL access does not know about the index and will not try to use it. The result may be a slow query, but the query eventually returns the correct results.

How to Access

In Data Sources Explorer, right-click the name of a database and select **Check Database**, or right-click a SQL table name under either the **Data Paths** or **SQL Tables** node, then select **Check Tables**. You can select multiple tables by holding down the Shift or Control key then clicking the desired table names.

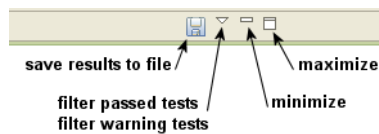
DBCheck (consistency check results) View



Features

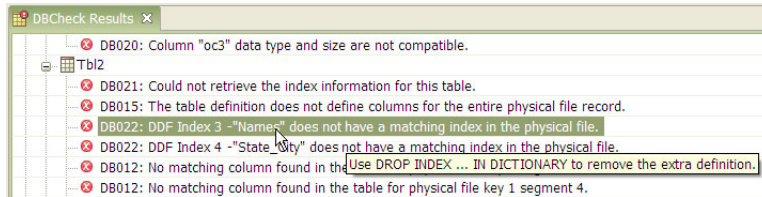
You can undock the DBCheck pane and move it to another place in the DDF Builder window. The pane can be minimized and maximized.

The pane also provides icons for saving the consistency check results to a text file and for hiding validation check messages and warnings by filtering them.



Error Message Tips

A tip appears if you click on an error message. The tip provides comments about the message and may include a suggested corrective action. The tips are also written to the text file if you save the DBCheck results to a file.



Copy SQL Definition

The Copy SQL Definition Wizard creates a new SQL table by using the schema of an existing SQL table as its source. The wizard also creates the Btrieve file associated with the new SQL table.

How to Access

In Data Sources Explorer, right-click a SQL file name and select **Copy SQL Definition**.

Features

The wizard requires that you specify the following:

- Server on which the database engine is running.
- Database in which you want to include the new table.
- Name of the new table. By default, this name becomes the name of the associated Btrieve file. You cannot use an existing table name for the copy.

Note: The new table does not include data, but you can export it from the source table and then import it into the new table. See [Importing Data with Import Data Wizard](#) and [Exporting Data with Export Data Wizard](#).

Export Btrieve Schema

The Export Btrieve Schema Wizard creates an XML file that specifies the schema of a source Btrieve file. You can use this XML file to create a new Btrieve file based on the structure of an existing Btrieve file. See [Import Btrieve Schema](#).

Data in the source file is not exported. If you want to export data, see [Exporting Data with Export Data Wizard](#).

How to Access

In Data Sources Explorer, right-click a Btrieve file name under the **Data Paths** node, then click **Export Btrieve Schema**.

Features

By default, the wizard uses the source file name for the exported XML file name. The wizard adds the file extension .xml and writes the output file to the same directory as the source file. You can change the default name and location of the output file.

The wizard also provides a preview of the XML content before you export it.

Import Btrieve Schema

The Import Btrieve Schema Wizard creates a Btrieve file based on the structure of another Btrieve file. The structure must be a schema of the source file in XML format.

Data from the source file is not imported. If you want to import data, see [Importing Data with Import Data Wizard](#).

How to Access

In Data Sources Explorer, right-click a data path under the **Data Paths** node, then select **Import Btrieve Schema**.

A Btrieve file must be located on physical storage, which is why the wizard is invoked from a data path.

Features

Since the wizard is invoked from the context of a known data path, you do not specify a path for the target file, only for a file name. The file name length must be less than or equal to 255 bytes. The name cannot contain spaces unless the Embedded Spaces client configuration option is enabled. The option is enabled by default. See [Embedded Spaces](#) under Application Characteristics properties for clients in *Advanced Operations Guide*.

Add Data Path

Data paths are added to the Data Sources Explorer by using **Add Data Path**. A data path represents a location in physical storage where Btrieve files reside. Each database must have at least one data path identified for it.

DDFs that you create for Btrieve files are located in the original data path for the database because all SQL tables for an entire database are defined in the same set of DDFs.

How to Access

In Data Sources Explorer, right-click the **Data Paths** node, then click **Add Data Path**.

Features

The existing directory can be empty or contain files.

Use the Delete command to remove a data path from Data Sources Explorer (right-click **Data Paths** or a database name, then click **Delete**). The directory is *not* deleted from physical storage.

Change Associated Data File

DDF Builder allows you to change the data file associated with a selected table definition (SQL Table).

How to Access

In Data Sources Explorer, right-click the SQL Table which you want to change the associated data file, then click **Change Associated Data File**. You may enter or browse to select the full path name for the data file you want associated.

Features

When changing the data file associated with a particular SQL table, you may enter the filename or use the browse button to select a file, based on the location.

Btrieve Types

DDF Builder provides a separate view for displaying the Btrieve data types and sizes, along with the corresponding SQL data types to which they map.

The Btrieve Types view can be used to analyze data. When you highlight bytes in the raw data view, the Btrieve Types view displays the data as it would appear for data types compatible with the size. When a particular column is selected in the Table Definition Editor, the Btrieve Types view also displays a preview of the formatted data.

How to Access

The Btrieve Types tab is located in the left pane of the DDF Builder window, next to the Data Sources Explorer tab. Click the **Btrieve Types** tab to display the view.

Btrieve Type	Size	SQL Type	Preview
Auto Increment	2	SMALLIDENTITY	
Auto Increment	4	IDENTITY	129370377
Bfloat	4	BFLOAT4	
Bfloat	8	BFLOAT8	
Bit	1	BIT	
Blob	8	LONGVARBINARY	
Clob	8	LONGVARCHAR	
Constant String		CHAR	□□□□
Currency	8	CURRENCY	
Date	4	DATE	09/09/1...
Date Time	8	DATETIME	
Decimal		DECIMAL	9091160
Float	4	REAL	2.73896...
Float	8	DOUBLE	
Guid	16	UNIQUEIDENTI...	
Integer	1	TINYINT	
Integer	2	SMALLINT	
Integer	4	INTEGER	129370377
Integer	8	BIGINT	
Logical	1	BIT	
Logical	2	SMALLINT	
Lstring		CHAR	□□□□
Money		MONEY	9101.6
Numeric		NUMERIC	9967.0
Numeric SA		NUMERICSA	9967.0
Numeric STS		NUMERICSTS	996.0
String		CHAR	□□□□
Time	4	TIME	05:46:0...
Tstamp	8	TIMESTAMP	
Unsigned Binary	1	UTINYINT	
Unsigned Binary	2	USMALLINT	
Unsigned Binary	4	UIINTEGER	129370377
Unsigned Binary	8	UBIGINT	
Wstring		CHAR	□□
Wzstring		CHAR	□□
Zstring		VARCHAR	□□□□

Features

The Btrieve Types pane is useful when you are trying to resolve incorrect data type and size conflicts by comparing and previewing data. To support this task, you can undock the Btrieve Types pane and move it to a more convenient place in the DDF Builder window.

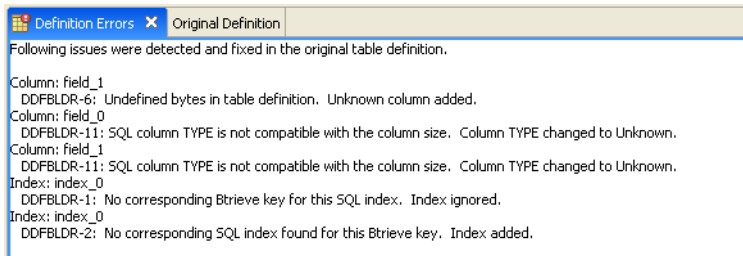
A preview column is also provided, showing how the selected data would appear as that particular data type. Previewing the data in each column can also be used to determine the size of a data type.

Definition Errors

Anytime that DDF Builder detects problems and alters your existing table definition, the Definition Errors window automatically opens. If you close this window, you can later review the changes by opening it.

How to Access

In the DDF Builder menu, select **Window > View Definition Errors**.



If no errors have occurred, the menu command is grayed out.

Features

The issues listed in the Definition Errors window give specific information regarding the problems addressed by DDF Builder. The original table definition is also provided in a read-only mode in the Original Definitions pane. You can open these panes side-by-side to make it easier to compare them.

Definition Errors List

The following table lists the possible table definition errors that can be detected by DDF Builder in your existing definitions. A general description of what caused the error and how DDF Builder has modified the table definitions as a result are also provided.

The Message Displays...	What it means...
DDFBLDR-1: No corresponding Btrieve key for this SQL index. Index ignored.	The Btrieve file does not contain a key to correspond with a SQL index in the existing table definition. The index in the SQL table is ignored.

The Message Displays...	What it means...
DDFBLDR-2: No corresponding SQL index found for this Btrieve key. Index added.	The Btrieve file contains a key that is not defined as a SQL index in the existing table definition. An index is added to the table definition that corresponds with the key in the Btrieve file.
DDFBLDR-3: SQL segment null flag does not match corresponding Btrieve segment null flag.	The Btrieve file contains a segment with a null flag that is not present in your SQL file segment. OR The SQL file contains a segment with a null flag that is not present in your Btrieve file.
DDFBLDR-4: SQL column overlaps another column. The column's size has been truncated.	The existing table definition contains a SQL column that overlaps another column in the table.
DDFBLDR-5: No corresponding SQL column found for Btrieve segment. New column added.	The Btrieve file contains a segment that has no corresponding SQL column in the existing table definition.
DDFBLDR-6: Undefined bytes in table definition. Unknown column added.	The existing table definition does not account for the same number of bytes found in your Btrieve file. The table definition contains some undefined bytes. An new unknown column is added to the table definition to account for the undefined bytes in your Btrieve file.
DDFBLDR-7: SQL column type does not match Btrieve segment type. Column type changed.	The Btrieve file segment uses a different type than what was found in your SQL table column. The data type in your SQL table column is changed to the same data type as the Btrieve file segment.
DDFBLDR-8: SQL column null flag does not match Btrieve segment null flag. Column null flag changed.	The existing table definition contains a SQL column with a different null flag setting than what DDF Builder found in the corresponding Btrieve segment.
DDFBLDR-9: SQL column case flag does not match Btrieve segment case flag. Column case flag changed.	The existing table definition contains a SQL column with a different case flag setting than what DDF Builder found in the corresponding Btrieve segment.

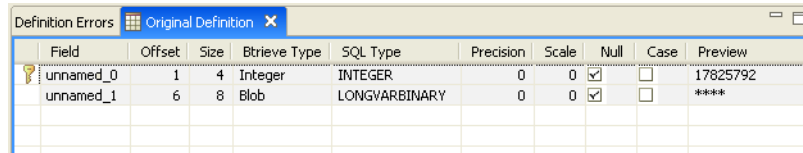
The Message Displays...	What it means...
DDFBLDR-10: SQL column defined across a Btrieve segment boundary. Column ignored.	<p>The existing table definition contains a SQL column defined across the boundary of a segment in your Btrieve file.</p> <p>The SQL column defined across the segment boundary is ignored in the existing table definition.</p>
DDFBLDR-11: SQL column type is not compatible with the column size. Column type changed to Unknown.	<p>The existing table definition contains a SQL column with a data type that is not compatible with the column size.</p> <p>The SQL column type is changed to unknown; size is not changed.</p>
DDFBLDR-12: Variable length column can only be the last column in a table. Column type changed to Unknown.	<p>The existing table definition defines a variable length record portion in a column that is not last in the table.</p> <p>The SQL column type is changed to unknown.</p>
DDFBLDR-13: Invalid bit mask detected. Bit mask value corrected.	<p>The table definition contains incorrect bit masks that do not equal the sizes needed to complete the byte. The bit masks have been changed in the table definition so their sizes are correct and total the byte.</p>
DDFBLDR-14: No compatible SQL type for Btrieve WSTRING or WZSTRING. Column type set to SQL CHAR. Column not usable.	<p>The Btrieve file contains columns defined with the WSTRING or WZSTRING type. These types do not map to a SQL data type. These columns are not usable and have been changed to CHAR type.</p>

Original Definition

Anytime that DDF Builder detects problems and alters an existing table definition, its original table definition is retained until the changes are saved. If you make changes to an existing table definition, the original definition is also retained until your changes are saved. The original table definition can be viewed by opening the Original Definition view.

How to Access

In the DDF Builder menu, select **Window > View Original Definition**.



Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
unnamed_0	1	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	17825792
unnamed_1	6	8	Blob	LONGVARBINARY	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	****

If no changes have occurred, the menu command is grayed out.

Features

The original table definition is provided in a read-only mode in this tab. This window may be moved so that comparisons to the updated table definition may be viewed.

You can always retain the original table definition by saving either the modified or the original definition with a different name. To do this, click **File > Save As** and enter a different name.

DDF Builder Tasks

This section explains the tasks that you can accomplish with DDF Builder. They are divided into the following categories:

Category	Description
General Tasks	Orient you to the overall use of DDF Builder
Tasks Initiated from Data Sources Explorer	Allow you to access editors, views, and wizards

General Tasks

General tasks apply to the overall use of the tool.

To start DDF Builder

See [Starting DDF Builder](#)

Accessing User Documentation

To access DDF Builder online help

1. Start DDF Builder
2. Click the part of the user interface that interests you
3. Press **F1**.
4. If you click another area of the user interface, you may need to press **F1** again to refresh the help view.

You can also select **Help > Zen Documentation Library**. Because DDF Builder and Zen Control Center (ZenCC) share common components, such as SQL Editor, online help opened within DDF Builder may also display information about other Zen features.

DDF Builder Log File

To access DDF Builder Log file

1. Start DDF Builder and click **Help > DDF Builder Log**.

The log file opens in your system text editor.

To clear DDF Builder Log file

1. Start DDF Builder and click **Help > Clear DDF Builder Log**.

A dialog box opens for you to confirm the action.

2. Click **Yes** to confirm and the log file is deleted.

The next time you start DDF Builder, a new log file is automatically created.

To see added or deleted nodes in Data Sources Explorer

1. Right-click the node for which you want to see the addition or deletion.
2. Click **Refresh**.

The context of the Refresh command applies to the node from which you invoke it. If you want to refresh *all* nodes in Data Sources Explorer, execute the command from the **Engines** node (the top of the tree).

Tasks Initiated from Data Sources Explorer

Most tasks in DDF Builder are initiated by selecting a command relevant to a node in Data Sources Explorer. The commands invoke an editor, a view, or a wizard. The following tables direct you to tasks based on what action you want to take. The tables are categorized by areas of interest:





- [Retrieve Files](#)
- [Data](#)
- [Database](#)
- [Data Sources Explorer](#)
- [Online Help](#)
- [SQL Queries](#)
- [SQL Tables](#)

Note: When using DDF Builder with security-enabled databases, you should always take the database offline and turn off all security prior to working with the files in DDF Builder.




Btrieve Files

If You Want To...	Then...	Notes
Create a new Btrieve file with an editor.	Right-click a data path under the Data Paths node. Click Create Btrieve File  .	DDF Builder does <i>not</i> automatically create DDFs for the new file. See Btrieve File Editor .
Export the Btrieve file structure to an XML file.	Right-click a Btrieve file name under the Data Paths node. Click Export Btrieve Schema  .	The data in the Btrieve file is <i>not</i> exported. See Export Btrieve Schema .
Create a new file based on the structure of another Btrieve file, as specified in a XML file.	Right-click a data path under the Data Paths node. Click Import Btrieve Schema  .	The data in the Btrieve file is <i>not</i> imported. See Import Btrieve Schema .
Specify a directory where you want a Btrieve file located.	Right-click the Data Paths folder  .	Any DDFs that you create for the Btrieve file are located in the original data path for the database. This is because all SQL tables are defined in the same set of DDFs. See Add Data Path .
Remove a data path directory from Data Sources Explorer.	Right-click a data path under the Data Paths node. Click Delete  .	Neither the directory nor any files in it are deleted from physical storage.
View the Btrieve file statistics with which the SQL table is associated.	Right-click a SQL table name under either the Data Paths or SQL Tables node. Click Edit Table Definition  .	The table names appear under the SQL Tables node, or under Data Paths node. See Statistics Page .
Change the Btrieve (data) file associated with the selected SQL table.	Right-click a SQL table name under either the Data Paths or SQL Tables node. Click Change Associated Data File  .	Only one data file may be associated with a particular table.


Data

If You Want To...	Then...	Notes
Check the consistency for all tables at once	Right-click Databases  node . Click Check Database .	See Check Table Consistency .
Check the consistency for one or more tables individually	Right-click a SQL table name under either the Data Paths or SQL Tables node. Click Check Tables .	The names appear under the SQL Tables node, or under a Btrieve file name (Data Paths node) if the Btrieve file has DDFs. You can select multiple tables by holding down the Shift or Control key then clicking the desired table names. See Check Table Consistency .
Specify in Data Sources Explorer a directory where you want data files to reside or where they already reside	Right-click the Data Paths folder  . Click Add Data Path  .	See Add Data Path .
Remove from Data Sources Explorer a directory where data files reside	Right-click a data path listed under the Data Paths node. Click Delete  .	Neither the directory nor any files in it are deleted from physical storage.

Database

If You Want To...	Then...	Notes
Create a new database	Right-click Databases  node . Click Add Database .	See Add Database .
Remove a database from Data Sources Explorer	Right-click Databases  node . Click Delete  .	The database is not deleted from physical storage.


Data Sources Explorer

If You Want To...	Then...	Notes
Update the tree to show additions to or deletions from nodes	Right-click any node except for the name of a SQL table. Either click Refresh  or press F5 .	The context of the Refresh command applies to the node from which you invoke it. If you want to refresh <i>all</i> nodes in Data Sources Explorer, execute the command from the Engines node at the top of the tree.

Online Help

If You Want To...	Then...	Notes
Access the user documentation	Press F1 (or Shift F1 for Linux) within an editor, view, or wizard (or click Help > DDF Builder Help Contents).	See the Note in Accessing User Documentation .

SQL Queries

If You Want To...	Then...	Notes
Execute a SELECT statement for all records in a SQL table (SELECT * FROM) or Access an editor in which you can type SQL statements to execute against a SQL table	Right-click a SQL table name under either the Data Paths or SQL Tables node. Click SQL Editor  .	By default, SQL Editor executes a SELECT * FROM statement when it opens. SQL table names display as subordinate nodes under SQL Tables and under the Btrieve file name (provided the Btrieve file has DDFs). See Add Database .
Execute SQL statements in SQL Editor or in Outline View	See Execution Tasks in <i>Zen User's Guide</i> .	You can perform numerous tasks with SQL Editor. See SQL Editor Tasks in <i>Zen User's Guide</i> for a complete list.

SQL Tables

If You Want To...	Then...	Notes
Create a new table definition with an editor	Right-click a Btrieve file name under the Data Paths node. Click Create Table Definition  .	You can create one or more SQL tables for a Btrieve file. Each SQL table appears as a subordinate node under the Btrieve file name. See Table Definition Editor .
Create a new table based on the schema of another SQL table	Right-click a SQL table name under either the Btrieve file or SQL Tables node. Click Copy SQL Definition  .	The table names appear under the SQL Tables node, or under a Btrieve file name (Data Paths node) if the Btrieve file has DDFs. See Copy SQL Definition .
Modify a table definition	Right-click a SQL table name under either the Data Paths or SQL Tables node. Click Edit Table Definition  .	The names appear under the SQL Tables node, or under a Btrieve file name (Data Paths node) if the Btrieve file has DDFs. See Table Definition Editor .
Remove a table from Data Sources Explorer	Right-click a SQL table name under either the Data Paths or SQL Tables node. Click Delete  .	The table is not deleted from physical storage. The table names appear under the SQL Tables node, or under a Btrieve file name (Data Paths node) if the Btrieve file has DDFs.
Change the Btrieve (data) file associated with the SQL table.	Right-click a SQL table name under either the Data Paths or SQL Tables node. Click Change Associated Data File  .	Only one data file may be associated with a particular table.
View the Btrieve file statistics with which the SQL table is associated	Right-click a SQL table name under either the Data Paths or SQL Tables node. Click Edit Table Definition  .	The table names appear under the SQL Tables node, or under Data Paths node. See Statistics Page .



DDF Builder Tutorials

The following tutorials provide learning materials for using DDF Builder:

- [Using the DDF Builder Tutorials](#)
- [Tutorial 1 – Creating Table Definitions with DDF Builder](#)
- [Tutorial 2 – Modifying Table Definitions with DDF Builder](#)
 - [Lesson 1 – Working with v3.00 DDFs](#)
 - [Lesson 2 – Working with Pre v6.x File Formats](#)
 - [Lesson 3 – Invalid Data Types and Sizes](#)
 - [Lesson 4 – Overlapping Column Definitions](#)
 - [Lesson 5 – File/Field Flag Inconsistencies](#)
 - [Lesson 6 – Index Inconsistencies](#)
 - [Lesson 7 – Variable Length Record Mismatch](#)
 - [Lesson 8 – Record Length Mismatch](#)

Using the DDF Builder Tutorials

This chapter provides you with two tutorials for using DDF Builder. Each tutorial provides you with basic information as to the structure of the data, a sample scenario in which the file is used, the goal of the tutorial, and the general steps needed to achieve the goals for each exercise.

Note: Whether you are creating or modifying table definitions with DDF Builder, you must have some knowledge about the structure of your data. It is helpful if you know the column definitions, as well as the offsets and sizes of the fields in the Btrieve file. DDF Builder attempts to guide you with the construction of column definitions, but there are limitations as to how much guidance it can provide.

Tutorial 1 Overview

The first tutorial addresses the situation where you have a Btrieve file without the necessary table definitions needed to provide relational access. Since there are no DDFs for the Btrieve file, you first need to create DDFs. This tutorial steps you through creating a new database with empty DDFs and adding a table definition to them for the Btrieve file.

Working through this tutorial also provides you with some fundamental knowledge for working with the DDF Builder interface. If this is your first time to work with DDF Builder you may want to first become familiar with the interface and where components are located by reviewing Chapter , [Using DDF Builder](#).

Tutorial 2 Overview

The second tutorial addresses a bit more complex situation where you modify existing table definitions because they contain outdated or incorrect table definition information. This tutorial steps you through reviewing the existing table definitions and making necessary changes to the DDFs associated with a set of Btrieve files.

The second tutorial is made up of several files and lessons, each focusing on a different issue that you may encounter when trying to modify DDFs using DDF Builder. Some of the issues are automatically addressed by DDF Builder; some require you to make the modifications manually.

Before You Begin

There are only a few things you need to do to start using these tutorials. This section reviews what you need to do to begin using the tutorials in this chapter.

Back Up Your Files

Creating or modifying table definitions with DDF Builder alters the structure of your database. As a precaution, you should always back up any files with which you intend to work. This includes both data files and any existing dictionary files.

Although opening your Btrieve files with DDF Builder only makes modifications to your DDFs and not your existing Btrieve files, it is still a good idea to backup all of your files or work from a copy of them. Existing table definitions could potentially be corrupted by a partial edit of the DDFs, leaving your definitions broken and unusable.

Note: If you intend to make modifications to existing Btrieve files *and* your DDFs that provide relational access, DDF Builder is not the utility to use. Zen Control Center allows you to change your DDFs and Btrieve files.

DDF Builder uses IN DICTIONARY calls to write table definitions to the DDFs. The Btrieve file is never written to when using DDF Builder.

Once you have backed up your data files or created a copy from which to work, you are ready to proceed with locating the files used by the tutorials.

Locate the Tutorial Files

The DDF Builder installation created the folders and files on your system for using the tutorials included in this chapter. The tutorial files are installed in the following locations from the default Application Data directory.

<Application Data>\DDFBuilder	DDF Builder Application Files
<Application Data>\DDFBuilder\tutorials\tutorial1	Tutorial 1 Files
<Application Data>\DDFBuilder\tutorials\tutorial2\v3	Tutorial 2 Files (Lessons 1 and 2)
<Application Data>\DDFBuilder\tutorials\tutorial2	Tutorial 2 Files (Lessons 3 through 8)

Note: For more information on the default Zen installation directories, see [Where are the files installed?](#) of *Getting Started with Zen*.

Create Data Source Names (DSN)

In order to access the tutorial databases within DDF Builder, you will need to make certain that each database has an associated Data Source Name (DSN) created.

Tutorial 1 walks you through creating a DSN at the time you create the database. This is because in Tutorial 1, you only have a Btrieve file with which to begin, so the database, DDFs and DSN all have to be created from scratch.

Tutorial 2 requires that you create two DSNs in order to see the files and folders within the DDF Builder interface. One DSN points to the Tutorial 2 database and the other points to the V3 database located in the Tutorial2 folder.

The following table lists the options for creating the Tutorial 2 and V3 database DSNs in ODBC Data Source Administrator.

Data Source Name	Description	Database Name
Tutorial2	ODBC Access	Tutorial2
V3	ODBC Access	V3

Tip... If you need help creating a DSN using the ODBC Data Source Administrator, see the *Zen User's Guide* for assistance.

Now that you have backed up your files, located the tutorial files, and created your DSNs, you are ready to start Tutorial 1.

Tutorial 1 – Creating Table Definitions with DDF Builder

Scenario

In this tutorial you have a Btrieve data file for which you need to provide ODBC access to create reports.

To do this, a table definition needs to be created for the file. Table definitions are stored in DDF files of the database. To your knowledge, there are no table definitions for this file, so you are starting from scratch.

Goals

The goal of this tutorial is to walk you through the steps necessary to provide relational access to a Btrieve file without any table definitions. To achieve the goal for this tutorial, you perform the following tasks using DDF Builder:

1. Create a Zen database
2. Open an Existing Btrieve file
3. Review DDF Builder's Findings
4. Define the Record Fields
5. Save Your Table Definitions
6. Review the Index Information
7. Preview the Defined Data

Tip... As you step through creating table definitions, you learn fundamental tasks for working with DDF Builder. This tutorial is designed to call out those tasks within these instructions so that you can become familiar with the common tasks for using DDF Builder.

What You Need to Know

Any time you work with DDF Builder to create or modify table definitions, you need some knowledge as to the structure of the data. Some of this information is stored in the Btrieve file itself, such as record length and index information. The column information, however, is not stored in the Btrieve file. The column information may have been provided by the software

vendor or the application developer. You can still attempt to create or modify table definitions by careful inspection of the data, but you should not use this utility without significant knowledge about the structure of your data.

For this tutorial, you know the data structure. Take a look at the structure for this file.

The Btrieve data file named CREATE_NEW.MKD has a record length of 110 bytes and contains six fields. Its data structure is defined as follows:

Field	Size	Data Type	Precision	Scale	Null	Case	Index
ID	4	Int			N		Y
First Name	20	String			Y	Y	
Last Name	20	String			N	Y	
DOB	4	Date			Y		
Address	50	String			Y	Y	
Income	8	Currency	8	2	Y		

The file contains a nonduplicatable key (unique index) set on the ID field; the index is named `indx_id`.

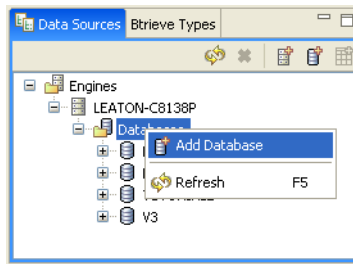
Now that you understand the data structure, you can get started. Begin by creating an empty database using DDF Builder, which provides a set of empty DDFs to define for the Btrieve file.

Create a Zen Database

Before you can create DDFs for an existing Btrieve file, you must first create a database. Begin by starting DDF Builder if it is not already running. See [Starting DDF Builder](#).

To create a database

1. In the Data Sources Explorer, expand the tree for the machine where you are creating the database. This should be the same machine where DDF Builder is installed.
2. Right-click the **Databases** icon and click **Add Database**.



The New Database Wizard opens for you to create a new database.

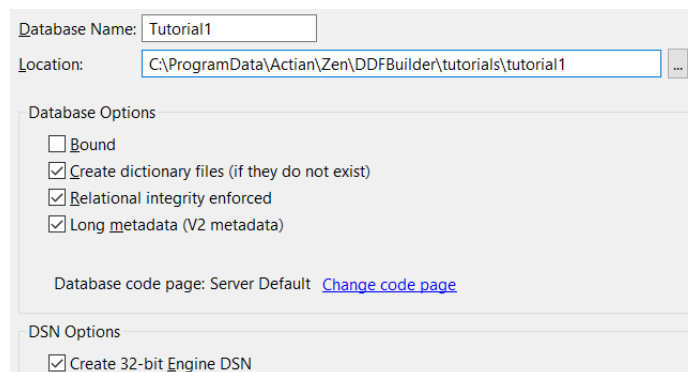
3. Complete the New Database Wizard using the following parameters:

- Database Name: Tutorial1
- Location:

`<Application Data>\DDFBuilder\tutorials\tutorial1\`

- Bound: not checked
- Create dictionary files: checked
- Relational integrity enforced: checked
- Long Metadata (V2 Metadata): checked
- Database code page: Server Default
- Create 32-bit Engine DSN: checked

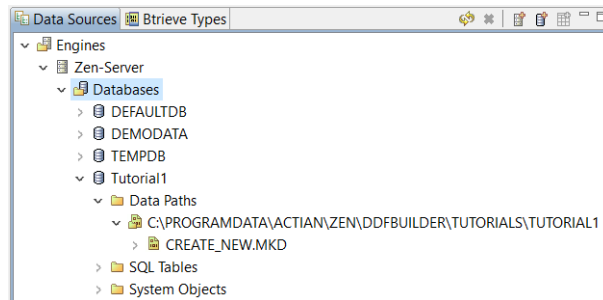
The New Database dialog should look similar to the following:



Tip... Creating the database in these steps also creates a Data Source Name (DSN). When you have finished working with the tutorials, you may want to delete the DSN and database created here.

4. Click **Finish** to create the database and the empty DDF files in the location you specified.

Once the database is created, it appears as a new node on the databases tree in the Data Sources Explorer.



The database node includes the Data Paths, SQL Tables, and System Objects folders. The Data Paths folder contains the locations of your Btrieve files. The SQL Tables folder contains any relational tables for the database, and the System Objects folder is the holder for your system dictionary tables, and in this case, empty data dictionary files.

Creating a Zen database using these parameters was the first step in creating table definitions for the Btrieve file. The database you just created contains dictionary files (DDFs), but these files currently contain only definitions for the structure of the DDFs themselves. Once you open the Btrieve file in DDF Builder, you can add a table definition to the DDFs created for this database so that the data in the Btrieve file can be accessed.

Notice that the database contains no SQL Tables – only the CREATE_NEW.MKD Btrieve file used for this tutorial. That is because there are no user tables defined. The empty DDFs are located in the System Objects folder, separate from the data files. As you create table definitions, you also will create corresponding SQL tables that will access the table definitions. Continue with opening the Btrieve file in DDF Builder.

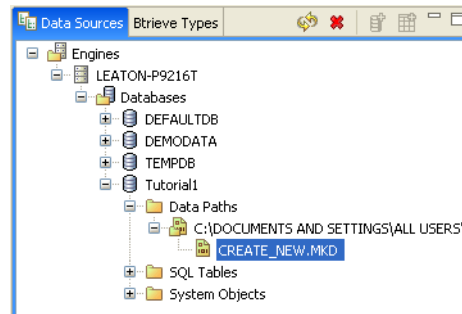
Open an Existing Btrieve File

Now that you have created DDF files in the same location as the Btrieve file, open the file in DDF Builder and see how the utility deciphers the data structure.

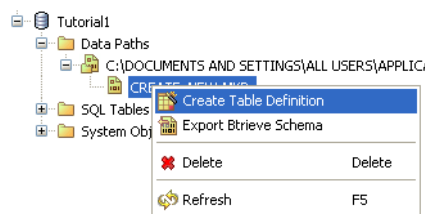
To open the Btrieve file

1. In DDF Builder, select the local machine and the database you just created.

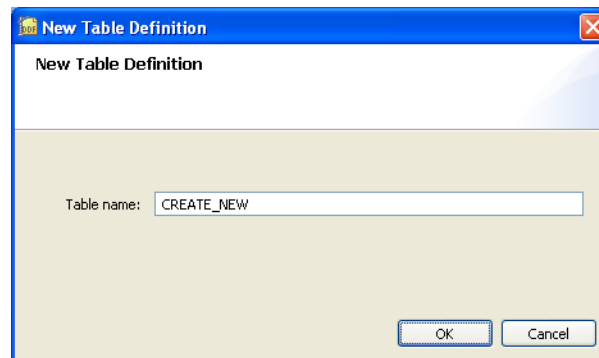
The Data Sources Explorer lists the Data Path and shows the directory location where the .mkd file was installed. This is the same location where you created the database. Your Tutorial1 database directory structure in the Data Sources Explorer should look similar to the following:



2. Right-click the CREATE_NEW.MKD folder containing the file, and select **Create Table Definition**.

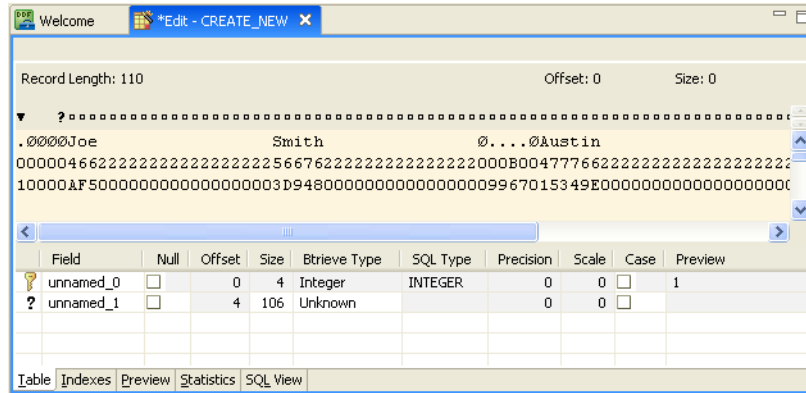


When you select to create the table definition, DDF Builder displays the New Table Definition dialog for you to associate a new table name with this set of table definitions. Think of this table as your corresponding relational table to your transactional file. When you have finished creating your table definitions, this SQL table should mirror the same structure as your original Btrieve file.



3. Enter CREATE_NEW as your Table name and click **OK**.

Once you provide a name for the table and click **OK**, DDF Builder begins to analyze the Btrieve file so that it can be opened. DDF Builder determines any known keys or indexes in the file and displays them as such inside the Table Definition Editor. Your display of the Table Definition Editor should look similar to the following:



Now that you have successfully opened the file, look at what DDF Builder found in more detail.

Review DDF Builder’s Findings

DDF Builder analyzes the data based on the file’s general statistics and the known keys or indexes in the file. In the file, DDF Builder detected a key in the first four bytes of the record and determined that the key is made up of an integer Btrieve data type. The key is illustrated in the Table Definition Editor with the key icon in the first column of the display.

Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
🔑 unnamed_0	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
? unnamed_1	<input type="checkbox"/>	4	106	Unknown		0	0	<input type="checkbox"/>	

The remaining parts of the record cannot be deciphered by DDF Builder, so they are grouped together and illustrated in the Table Definition Editor with the question mark icon and assigned a Btrieve type of Unknown.

Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
🔑 unnamed_0	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
? unnamed_1	<input type="checkbox"/>	4	106	Unknown		0	0	<input type="checkbox"/>	

Creating the table definitions for the Btrieve file involves splitting the unknown fields found by DDF Builder into the various fields you need – defining the particulars of each field as you go along.

Before you begin defining the record fields, you should make sure you understand how nulls are treated in the Table Definition Editor.

Note: This tutorial uses true nulls, but it is important to note that much older versions of Zen did not provide support for true nulls – only legacy nulls. True nulls were first introduced in Pervasive.SQL 2000. If you are working with files created prior to Pervasive.SQL 2000, you may want to skip the following section on nulls.

A Note About Nulls

When you define record fields in this tutorial, it is important to first make note if the field is nullable, since marking a section of the record as nullable adds an extra byte to the field. This is because in a Btrieve file, nullable portions of the record are designated by the use of the null indicator byte. In the raw data view of the Table Definition Editor, the null indicator byte is indicated in the byte immediately preceding the field or column. When the Null check box is selected, the null indicator byte becomes active and the size of the field is automatically reduced by one byte to accommodate for the null indicator byte.

Say you wanted to create a nullable field 50 bytes in size. You would select the field size as 51 bytes to accommodate for the null indicator byte. Once you select that the field is nullable, the size is automatically reduced to 50.

This information is helpful as you define the record fields.

Note: For more information on working with nulls in DDF Builder, see [Legacy Nulls in DDF Builder](#) and [True Nulls in DDF Builder](#).

Define the Record Fields

As you saw when you opened the Btrieve file, DDF Builder determined the index of four bytes and left an unknown field of 106 bytes. You define the unknown field into record fields.

To define the record fields

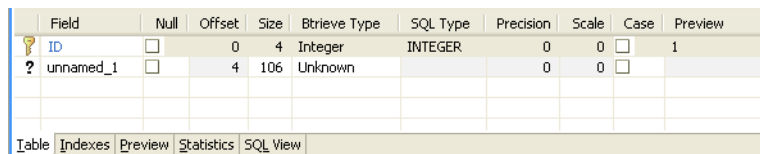
Although DDF Builder determined most of the attributes for the first field, a field name could not be determined, so go ahead and name the first field.

Name a Field

1. In the grid data view, select the unnamed_0 row, and enter the values needed to reflect the following information:
 - Field: ID
 - Null: not checked
 - Offset: 0
 - Size: 4
 - Btrieve Type: Integer
 - SQL Type: INTEGER
 - Precision: 0
 - Scale: 0
 - Case: not checked

DDF Builder determined a lot of the attributes for this row, so entering the field is the only change you will need to make. Be certain to verify all the values of the row.

Your screen should now reflect two rows in the grid data view – the key row you just named ID and the unknown _1 row found by DDF Builder.



Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
? unknown_1	<input type="checkbox"/>	4	106	Unknown		0	0	<input type="checkbox"/>	

Table | Indexes | Preview | Statistics | SQL View

Now you are ready to parse out and define the unknown data.

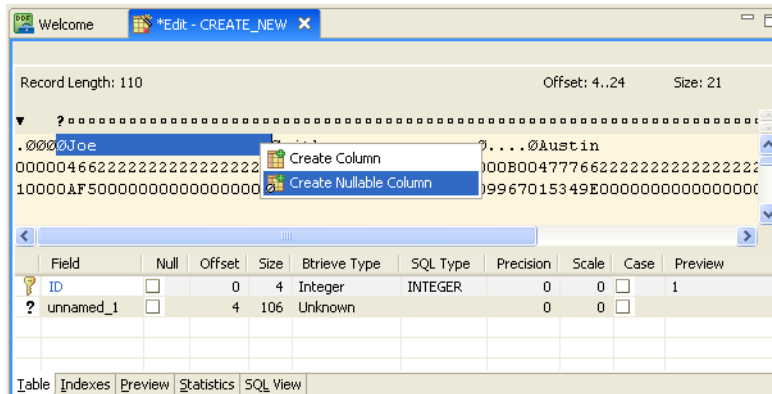
The next field to define is nullable. The following steps outline how to select data from the raw data view and create a nullable column from that data.

Create a Nullable Column in the raw data view

1. In the raw data view, beginning at offset 4, select 21 bytes of data.

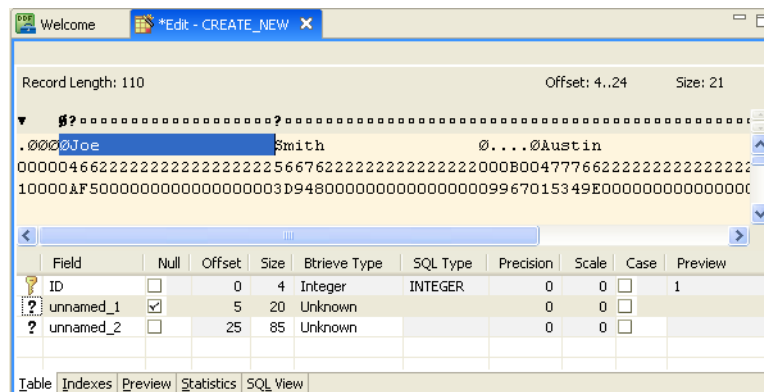
Tip... To find offset 4, locate the question mark just above the first record in the raw data view, or position the cursor until the Offset indicator shows 4.

2. Right-click and select **Create Nullable Column**.



Tip... Creating a column, whether in the raw data view or the grid data view, automatically creates a new column out of the remaining bytes in the record.

You may notice that although you selected 21 bytes, the grid data view shows the size as 20. This is because of the null indicator byte mentioned earlier. The null indicator byte is shown in the raw data view as the § symbol and accounts for the extra byte.



Now the editor shows three fields – the four byte integer (ID), the nullable column you just created (unnamed_1), and the new unknown field comprised of the remaining bytes (unnamed_2).

Before you go on, be sure and complete the definition for the nullable column you just created.

3. Select the unnamed_1 row in the grid data view.
4. Enter the values needed to reflect the following information:
 - Field: FirstName

- Null: checked
- Offset: 5
- Size: 20
- Btrieve Type: String
- SQL Type: Char(20)
- Precision: 0
- Scale: 0
- Case: checked

Tip... The number of bytes selected in the raw data view, along with creating the column as nullable, predetermined the size and null selections. The size also limits which data types you can select for the column.

Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
FirstName	<input checked="" type="checkbox"/>	5	20	String	CHAR(20)	0	0	<input type="checkbox"/>	Joe
? unnamed_2	<input type="checkbox"/>	25	85	Unknown		0	0	<input type="checkbox"/>	

Notice that the unknown field indicator is no longer displayed in the grid data view, once the Btrieve Type is set and the field is defined.

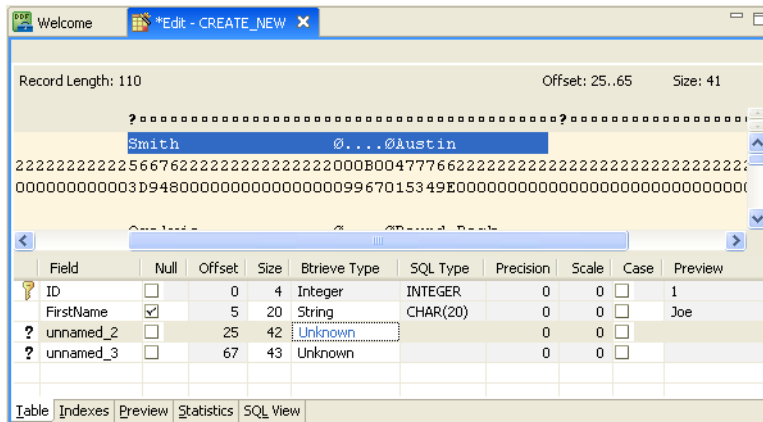
Continue defining the unknown fields in the file by splitting the unknown row into multiple columns within the grid data view.

Split a Column in the grid data view

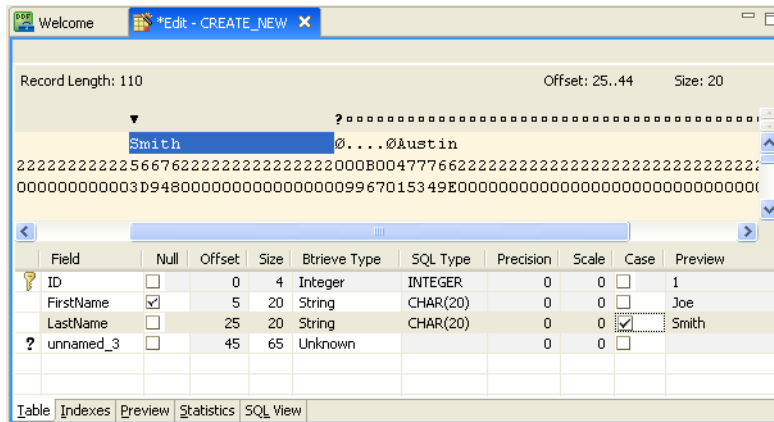
1. Select the unnamed_2 row in the grid data view.
2. Right-click and select **Split Column**.

Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
FirstName	<input checked="" type="checkbox"/>	5	20	String	CHAR(20)	0	0	<input type="checkbox"/>	Joe
? unnamed_2	<input type="checkbox"/>	25	85	Unknown		0	0	<input type="checkbox"/>	

When you split the column, the result is two columns of equal, or near equal, size. This means that the 85 byte column is now two columns, one with 42 and the other with 43 bytes.



3. Select the unnamed_2 row.
4. Enter the values needed to reflect the following information:
 - Field: LastName
 - Null: not checked
 - Offset: 25
 - Size: 20
 - Btrieve Type: String
 - SQL Type: Char(20)
 - Precision: 0
 - Scale: 0
 - Case: checked



When you change the size from 42 to 20, DDF Builder adds the extra 22 bytes to the unnamed_3 row.

You may have already noticed that the SQL Preview column displays the data as it is defined. This provides you a preview of how the data is interpreted by DDF Builder based on your table definitions.

- Right-click and select **Split Column** to split the unnamed_3 row.

Splitting the column changed unnamed_3 to 32 bytes and created unnamed_4 with 33 bytes.

Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
FirstName	<input checked="" type="checkbox"/>	5	20	String	CHAR(20)	0	0	<input type="checkbox"/>	Joe
LastName	<input type="checkbox"/>	25	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	Smith
? unnamed_3	<input type="checkbox"/>	45	32	Unknown		0	0	<input type="checkbox"/>	
? unnamed_4	<input type="checkbox"/>	77	33	Unknown		0	0	<input type="checkbox"/>	

Now define the unnamed_3; this field is nullable.

- Select the **Null** checkbox so that the null indicator byte is reserved at the first of the record.

Notice how selecting the Null checkbox automatically decremented the size to 31.

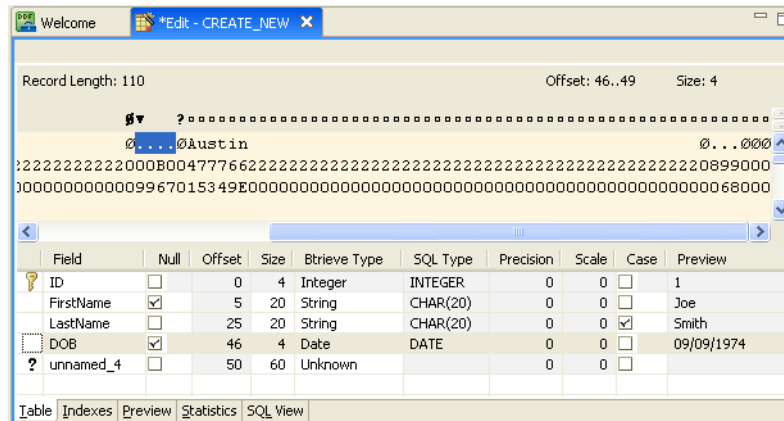
Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
FirstName	<input checked="" type="checkbox"/>	5	20	String	CHAR(20)	0	0	<input type="checkbox"/>	Joe
LastName	<input type="checkbox"/>	25	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	Smith
? unnamed_3	<input checked="" type="checkbox"/>	46	31	Unknown		0	0	<input type="checkbox"/>	
? unnamed_4	<input type="checkbox"/>	77	33	Unknown		0	0	<input type="checkbox"/>	

7. Select the unnamed_3 row.

8. Enter the values needed to reflect the following information:

- Field: DOB
- Null: checked
- Offset: 46
- Size: 4
- Btrieve Type: Date
- SQL Type: Date
- Precision: 0
- Scale: 0
- Case: not checked

Your table definition should now look similar to the following:



When you changed the size to 4, DDF Builder combined the remaining bytes with the next field.

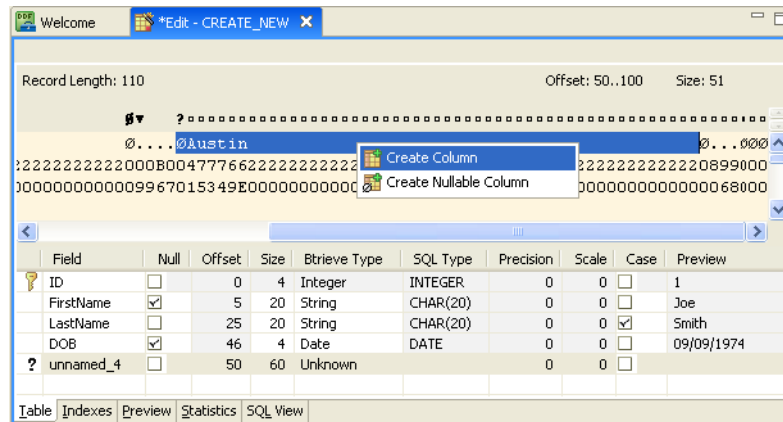
Now, there is one unknown field left with a size of 60 bytes. The data structure indicates that two more fields need defining. Use the raw data view to create the column.

Create a Column from the raw data view

1. Starting at offset 50, select 51 bytes in the raw data view.

Tip... To find offset 50, locate the question mark just above the first record in the raw data view, or position the cursor until the Offset indicator shows 50.

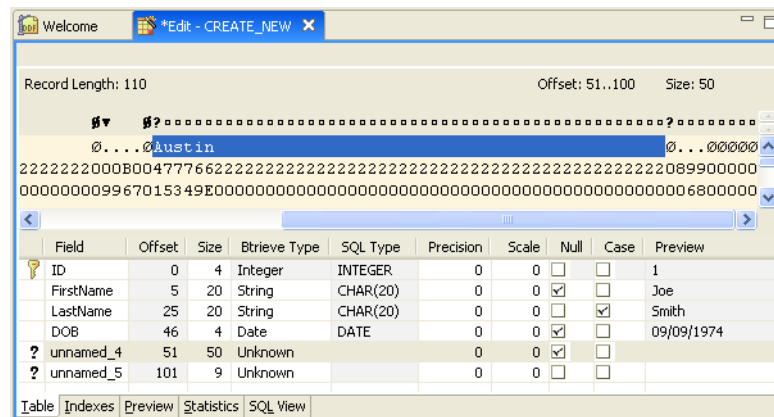
2. Right-click and select **Create Column**.



You created a not null column, but really you need a null column instead. Change the non-nullable column to nullable now using the grid data view.

Change a Non-Nullable Column to Nullable in the grid data view

1. Select the unnamed_4 row, and select the **Null** checkbox.



Checking the Null checkbox reserves the null byte indicator and reduces the size to 50 bytes.

2. Enter the rest of the values needed to reflect the following field information:

- Field: Address
- Null: checked
- Offset: 51

- Size: 50
- Btrieve Type: String
- SQL Type: Char(20)
- Case: checked

Your table definitions should now look similar to the following:

Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
FirstName	<input checked="" type="checkbox"/>	5	20	String	CHAR(20)	0	0	<input type="checkbox"/>	Joe
LastName	<input type="checkbox"/>	25	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	Smith
DOB	<input checked="" type="checkbox"/>	46	4	Date	DATE	0	0	<input type="checkbox"/>	09/09/1974
Address	<input checked="" type="checkbox"/>	51	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	Austin ...
? unnamed_5	<input type="checkbox"/>	101	9	Unknown			0	<input type="checkbox"/>	

Lastly, define the last field in the file.

3. Select unnamed_5 row in the grid data view, then make the following changes in this order:

- Null: checked
- Field: Income
- Btrieve Type: Currency

By selecting the Null checkbox first, you automatically set the size correctly. If you had first changed the size to 8, DDF Builder would have created a new unknown column from the last remaining byte.

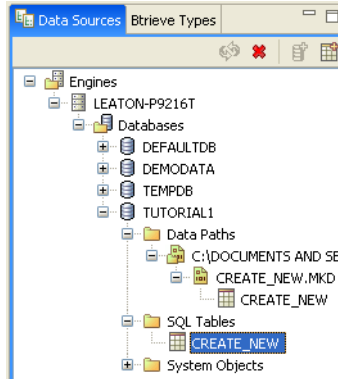
You have now finished defining the fields for your file. Your table definition should look like the following:

Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
FirstName	<input checked="" type="checkbox"/>	5	20	String	CHAR(20)	0	0	<input type="checkbox"/>	Joe
LastName	<input type="checkbox"/>	25	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	Smith
DOB	<input checked="" type="checkbox"/>	46	4	Date	DATE	0	0	<input type="checkbox"/>	09/09/1974
Address	<input checked="" type="checkbox"/>	51	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	Austin ...
Income	<input checked="" type="checkbox"/>	102	8	Currency	CURRENCY		0	<input type="checkbox"/>	1000.0

Before you proceed, be sure and save your table definition.

4. Select **File** > **Save**, or click the Save icon on the toolbar.

You have now successfully created your table definition in your DDFs. When you saved your work, SQL tables were created under the SQL Tables node of the Data Sources Explorer.

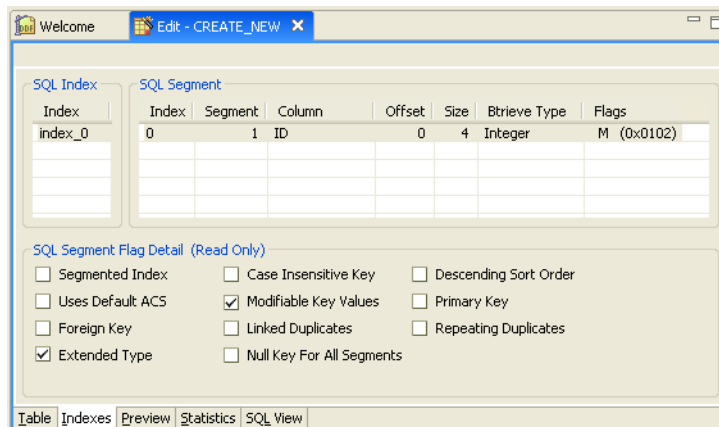


Now that you have created the table definitions, take a look at the index.

Review the Index Information

Now take a look at the indexes DDF Builder found in the file.

1. Click the **Indexes** tab in the Table Definition Editor.

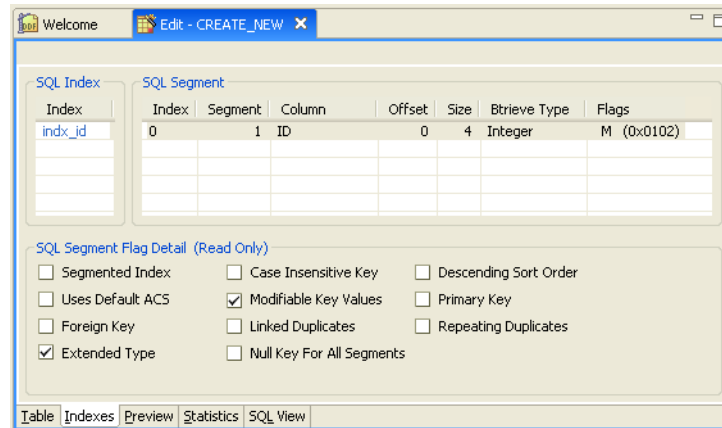


DDF Builder can detect any index created in the file, but because Btrieve does not use index names, the index has no associated name in DDF Builder.

Although the Indexes page is primarily read-only, you can enter or alter the index name.

Name an Index

1. Select the `index_0` entry in the Indexes column by double-clicking the field.
2. Enter `indx_id` as the name of the index from the data structure.



3. Before you proceed, be sure and save your index changes. Click **File** > **Save** to do so.

That is the extent of what DDF Builder allows you to do to Indexes.

You should always preview the data once you have completed your table definitions.

Preview the Defined Data

Next, take a moment to double-check the data in the file for which you have just created table definitions.

1. With DDF Builder still open, click the **Preview** tab in the Table Definition Editor.

Your Preview page should look like the following:

ID	FirstName	LastName	DOB	Address	Income
1	Joe	Smith	09/09/1974	Austin	1000.0
2	James	Ogelvie	09/29/1977	Round Rock	2000.0
3	Haniza	Yaacob	11/18/1961	San Antonio	3000.0
4	Harold	Dimbat	10/12/1974	Dallas	4000.0
5	Bradley	Giddy	03/19/1958	Houston	5000.0

The Preview page offers you a look at how the data in the file is formatted using the table definitions you just created. As you can see, all of the data appears reasonable for the column names and the type of data that should display.

The buttons at the bottom of this page allow you to move through the data in the file so that you can verify all of the records, or perhaps find data in an unexpected format that you may need to deal with to complete your table definitions.

If you find that some of your data does not display as you had expected, see next section, [Tutorial 2 – Modifying Table Definitions with DDF Builder](#).

If you make any changes to your already saved table definitions, be sure and save your work before closing DDF Builder.

Conclusion

Congratulations! You have completed Tutorial 1 and now have a set of DDFs that allow you to access the data in your Btrieve file using relational access.

The next tutorial deals with making modifications to a set of already created DDFs for a Btrieve file. The next tutorial also discusses some of the problems with DDFs that DDF Builder automatically detects.

Tip... Remember, this tutorial created a new database and a DSN. If you need to remove these from your system, you may want to do that now.

Tutorial 2 – Modifying Table Definitions with DDF Builder

Scenarios involving out-of-date or incorrect DDFs are numerous. For this reason, this tutorial is made up of several lessons to represent a range of situations with a Btrieve file or existing table definition. Each lesson is designed to address one problem so that you may begin to extrapolate the particulars of other situations.

This tutorial is organized in a way that allows you work through the exercises in each lesson, breaking them apart into manageable chunks of related information. You can also refer to this tutorial later as a kind of troubleshooting review to find a list of possible situations that you may encounter. From that list, you can proceed to the exact lesson that covers that situation and review possible solutions.

Scenario

You have a collection of Btrieve data files that have DDFs available, but the table definitions in the DDFs are outdated and need modifying. It may be a case of your receiving out-of-date files from the vendor, or the application was modified without making the proper updates to the DDFs.

Goals

The goal of this tutorial is to open all of the Btrieve files and existing table definitions. You need to review DDF Builder's findings, make any necessary changes and save the table definitions using DDF Builder. The table definitions should mirror the database schema and provide an accurate representation of the data in the Btrieve files.

The following lists lessons in this tutorial and the conditions they represent.

- [Lesson 1 – Working with v3.00 DDFs](#)
- [Lesson 2 – Working with Pre v6.x File Formats](#)
- [Lesson 3 – Invalid Data Types and Sizes](#)
- [Lesson 4 – Overlapping Column Definitions](#)
- [Lesson 5 – File/Field Flag Inconsistencies](#)
- [Lesson 6 – Index Inconsistencies](#)
- [Lesson 7 – Variable Length Record Mismatch](#)
- [Lesson 8 – Record Length Mismatch](#)

Lesson 1 – Working with v3.00 DDFs

Scenario

In this lesson, you have a database with an existing set of table definitions. The table definitions were created with a much older version of Zen and are no longer compatible with the version of Pervasive.SQL 2000 supported in this release of DDF Builder.

Caution! DDF Builder supports version 4.xx DDFs but does not support DDFs created prior to Scalable SQL version 4.xx.

Goals

The goal is to try and open the DDFs with DDF Builder. This tutorial explains how DDF Builder handles these files and provides you with a solution so that you can convert your files to a version compatible with DDF Builder.

What You Need to Know

The files to use for this lesson reside in a folder named V3. Assuming you installed using the default installation locations, this folder is located at:

```
<Application Data>\DDFBuilder\tutorials\tutorial12\v3
```

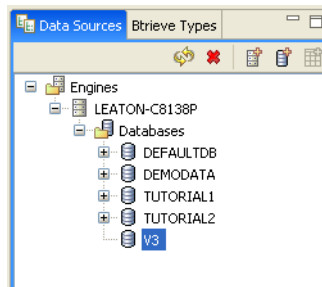
You must create a DSN for this database before continuing.

Note: Refer to [Create Data Source Names \(DSN\)](#) if you need information about creating the DSN.

Open the Btrieve File

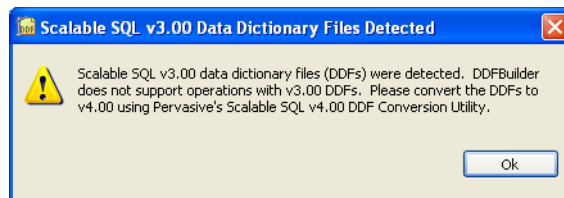
Begin by starting DDF Builder, if it is not already running.

1. In the DDF Builder Data Sources Explorer, expand the **Databases** node to see all of the available databases.
2. Locate the **V3** database in the list.



3. Double-click the **V3** database icon.

The following message opens:



4. Click **OK** to close the message window.

You are unable to open these files because they were originally created with a version of the product that is no longer supported.

Understanding the Warning Message

DDF Builder is unable to open Scalable SQL v3.00 data dictionary files because the format used at that time (Scalable SQL 3.01) is no longer supported.

Tip... DDF Builder is supported on PSQL v9 and later versions. DDFs created with Scalable SQL v4.00 and later are supported along with Btrieve data files v6.x and later.

How To Convert My Files

If you have older DDFs you need to convert them to the new format. Use the following steps.

1. Backup your existing, old database by copying the data files and DDFs to a different storage location.

-
2. Export the table schema for all tables in your old database using Zen Control Center (ZenCC). See [To export a database schema](#) in *Zen User's Guide*. Ensure that you select the export option “add IN DICTIONARY clause to CREATE statements.”
 3. Ensure that the CREATE INDEX statements in the export file put the index segments in the correct order for each table.

The indexes for the tables must be in the same order as in the original data files. The exported table schemas put the indexes in alphabetical order, so you may need to rearrange the statements.

To determine the sequential order of the indexes, you can use the `butil -stat` command against the original data file. See [Viewing Data File Statistics](#) in *Advanced Operations Guide*. Another method is to use the `dbo.fSQLStatistics` catalog function. See [dbo.fSQLStatistics](#) in *SQL Engine Reference*.

4. Typically, because you are converting old files, you need to add the following line to the export schema file as the *first* line in the file:

```
SET TRUENULLCREATE = OFF;
```

Add the following line as the *last* line of the file:

```
SET TRUENULLCREATE = ON;
```

5. Create a new database with ZenCC. See [To create a new database](#) in *Zen User's Guide*.
6. In ZenCC SQL Editor, execute the CREATE TABLE SQL statements in the export schema file against the new database to create all of the tables. See [To open a SQL script](#) in *Zen User's Guide*.
7. Copy the old Zen data files – not the DDFs – from the old database to the storage location for the new database data files.

You should be able to access your existing data via SQL (for example, with ZenCC).
8. Use DDF Builder to build table definitions for any Btrieve files that did not have a definition and to correct problem definitions. See also [Conversion Notes](#).
9. If required, rebuild your data files (it is likely they are a pre 6.x file format). For information on rebuilding your Btrieve data files, proceed to the next section, [Lesson 2 – Working with Pre v6.x File Formats](#).

Conversion Notes

The steps above are the most direct for simple data files. Try the steps first to determine the results of the conversion.

You may encounter two situations that require additional effort:

- [Alternating Collating Sequences](#)
- [Table Definitions That Require Using ZenCC and DDF Builder](#)

Alternating Collating Sequences

If a table column uses an alternating collating sequence (ACS), you must manually modify the CREATE TABLE statement. ZenCC does not include an ACS when you export a table schema.

To determine if a column uses an ACS, check table consistency with DDF Builder (see [Check Table Consistency](#)). If a message informs you of an attribute mismatch with physical key file segment, then the column uses an ACS.

The following example is a modified CREATE TABLE statement in which collating was added manually:

```
SET TRUENULLCREATE = OFF;
CREATE TABLE "PATAPP" IN DICTIONARY USING 'PATAPP.DTA' (
  "ID" CHAR(6) NOT NULL COLLATE 'UPPER.alt',
  "Appointment Date" DATE NOT NULL,
  "Appointment Time" TIME NOT NULL,
  "AMPM" CHAR(4) NOT NULL COLLATE 'UPPER.alt',
  "Doctor" CHAR(12) NOT NULL,
  "Code" CHAR(3) NOT NULL COLLATE 'UPPER.alt',
  "Amount Paid" MONEY(14,2) NOT NULL,
  "Date Paid" DATE NOT NULL);
CREATE INDEX "index_0" IN DICTIONARY ON "PATAPP" (
  "Appointment Date" ,
  "AMPM" ,
  "Appointment Time" );
CREATE INDEX "index_1" IN DICTIONARY ON "PATAPP" (
  "ID" );
CREATE INDEX "index_2" IN DICTIONARY ON "PATAPP" (
  "Code" );
SET TRUENULLCREATE = ON;
```

Also, if the indexes are not in the correct sequence for the table, you must rearrange the CREATE INDEX statements to put the index segments in the correct sequence.

Table Definitions That Require Using ZenCC and DDF Builder

Some table definitions may be so difficult to resolve that you have to use both ZenCC and DDF Builder. If columns are marked as “unknown” in DDF Builder, you may need to view the table in ZenCC Table Editor, then define the columns similarly in DDF Builder. Using both utilities and a back-and-forth method, you should be able to complete the table definitions.

Conclusion

This lesson introduced you to how DDF Builder handles Scalable SQL v3.01 DDFs and provided you with a solution for converting your files so that they may work with current versions of the software.

You also learned that databases created with Scalable SQL v3.01 DDFs are likely to contain Btrieve data files with a pre v6.x file format and must be rebuilt before using DDF Builder.

Lesson 2 – Working with Pre v6.x File Formats

Scenario

In this lesson, you have Btrieve files with an existing set of table definitions. Btrieve files in a pre v6.x format are no longer compatible with the version of Zen supported in this release of DDF Builder.

Caution! DDF Builder supports 6.x files but does not support files created before that version.

Goals

The goal is to try and open the files with DDF Builder. This tutorial explains how DDF Builder handles pre v6.x file formats and v6.x and later file formats. This tutorial also provides a solution to make the files compatible with DDF Builder.

What You Need to Know

The files to use for this lesson and the remaining lessons in this tutorial reside in a folder named Tutorial2. Assuming you installed using the default installation locations, this folder is located at:

```
<Application Data>\DDFBuilder\tutorials\tutorial2
```

You must create a DSN for this database before continuing.

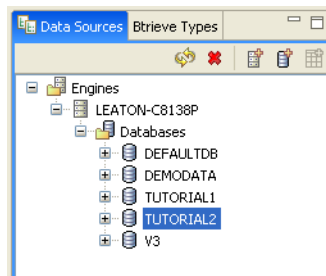
The files specific to this lesson are KO.BTR and KO.MKD.

Note: See [Create Data Source Names \(DSN\)](#) for information about creating the DSN.

Open the Btrieve File

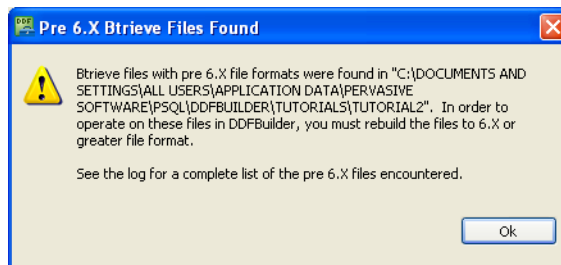
Begin by starting DDF Builder, if it is not already running.

1. In the DDF Builder Data Sources Explorer, expand the Databases node to see all of the available databases.
2. Locate the Tutorial2 database in the list.



3. Double-click the Tutorial2 database icon.

The following message appears:



4. Click **OK** to close the message window.

You are unable to open all of the files because some files in the folder are pre v6.x.

Understanding the Warning Message

DDF Builder is unable to open files that are version 5.x and earlier because that format is no longer fully supported. To use the pre v6.x files, you must rebuild the files to a 6.x version or greater file format.

Tip... DDF Builder is supported on PSQL v9 and later.

Before you rebuild the version 5.x and earlier files, check the log file to determine exactly which files you need to rebuild.

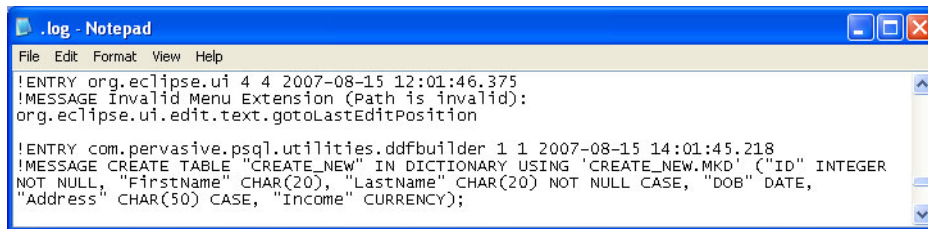
View the Log File

The files you need to rebuild are listed in the DDF Builder log file.

To access the log file

1. With DDF Builder running, click **Help**.
2. Click **DDF Builder Log File**.

The Log file opens in the default text editor on your system. The following is a sample from the log file for this lesson.

A screenshot of a Notepad window titled ".log - Notepad". The window contains two log entries. The first entry is: "!ENTRY org.eclipse.ui 4 4 2007-08-15 12:01:46.375 !MESSAGE Invalid Menu Extension (Path is invalid): org.eclipse.ui.edit.text.gotoLastEditPosition". The second entry is: "!ENTRY com.pervasive.psqli.utilities.ddfbuilder 1 1 2007-08-15 14:01:45.218 !MESSAGE CREATE TABLE 'CREATE_NEW' IN DICTIONARY USING 'CREATE_NEW.MKD' ('ID' INTEGER NOT NULL, 'FirstName' CHAR(20), 'LastName' CHAR(20) NOT NULL CASE, 'DOB' DATE, 'Address' CHAR(50) CASE, 'Income' CURRENCY);".

```
.log - Notepad
File Edit Format View Help
!ENTRY org.eclipse.ui 4 4 2007-08-15 12:01:46.375
!MESSAGE Invalid Menu Extension (Path is invalid):
org.eclipse.ui.edit.text.gotoLastEditPosition

!ENTRY com.pervasive.psqli.utilities.ddfbuilder 1 1 2007-08-15 14:01:45.218
!MESSAGE CREATE TABLE 'CREATE_NEW' IN DICTIONARY USING 'CREATE_NEW.MKD' ('ID' INTEGER
NOT NULL, 'FirstName' CHAR(20), 'LastName' CHAR(20) NOT NULL CASE, 'DOB' DATE,
'Address' CHAR(50) CASE, 'Income' CURRENCY);
```

Notice that the log file lists each pre v6.x file, including the full data path for the file. Using this data path, you can locate the files that need rebuilding.

How To Rebuild My Files

Use the Zen Rebuild Utility to rebuild your files to version 6.x or later. The Rebuild Utility is installed as one of the core utilities of Zen and is available from the Tools menu of the Zen Control Center.

Where To Go From Here

The Rebuild Utility is available from the Zen Control Center. For information on using this utility, see “Converting Data Files,” in the *Advanced Operations Guide*.

Conclusion

This lesson introduced you to how DDF Builder handles pre 6.x version file formats and provided you with a solution for rebuilding your files to work with them in DDF Builder.

Lesson 3 – Invalid Data Types and Sizes

Scenario

In this lesson, you have a Btrieve file with incorrect data types and sizes. As a result of this, the data does not format in a way that is understandable.

Goals

The goal is to open the file's existing table definitions with DDF Builder. You will inspect all the data types and sizes using the Btrieve Types view and make the modifications needed so that the data is formatted using the correct data type for the field size.

What You Need to Know

For this lesson, use the file named `Type_Size.MKD`. This file resides in a directory named `Tutorial2`. Assuming you installed using the default installation locations, this folder is located here:

```
<Application Data>\DDFBuilder\tutorials\tutorial2
```

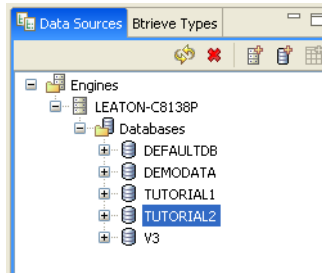
This directory is part of the `Tutorial2` database.

Note: You must have a Data Source Name (DSN) that points to this database in order to access the data in this tutorial. If you have not yet created this DSN, refer to [Create Data Source Names \(DSN\)](#).

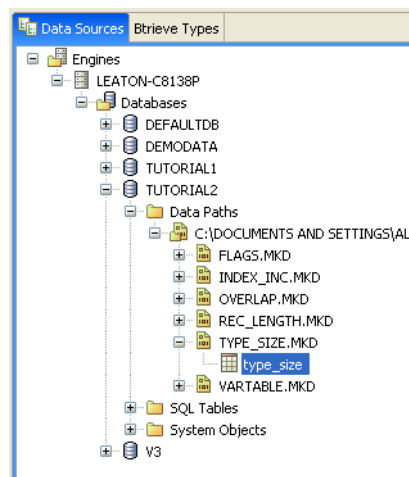
Open the Btrieve File

You should have DDF Builder already running from the last lesson. If not, begin by starting DDF Builder.

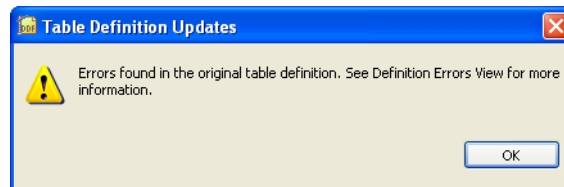
1. In the DDF Builder expand the Data Sources Explorer nodes, and locate the `Tutorial2` database in the list.



2. Double-click the Tutorial2 database icon to expand the nodes.
3. Expand the nodes until the Type_Size.MKD file and the associated SQL table display.



4. Double-click the type_size SQL table, or right-click and select **Edit Table Definition**.
The Table Definition Editor opens and displays the following message:



This message tells you that DDF Builder has analyzed your existing table definitions and found problems with those definitions. As a result of this, DDF Builder had to make some modifications to open and display the existing table definitions.

5. Click **OK** to clear the message and display the table definition.

Tip... For a complete list of possible definition errors, refer to [Definition Errors List](#).

Look for Inconsistencies

Begin by looking at the DDF Builder interface to review the inconsistencies with which you are dealing. Specifically, look at the grid data view and the Definition Errors window.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Unknown		0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	2	Unknown		0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
unnamed_6	53	2	Unknown		0	0	<input type="checkbox"/>	<input type="checkbox"/>	
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Table | Indexes | Preview | Statistics | SQL View

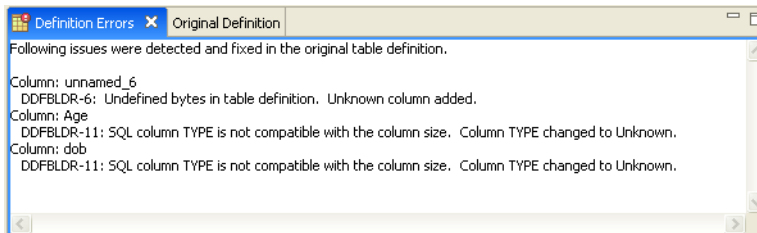
The grid data view of the Table Definition Editor shows your existing table definition with the modifications that DDF Builder made.

Note: DDF Builder changes are not automatically saved. Any modifications made by DDF Builder must be saved.

In this example, DDF Builder gives visual indications as to the columns that need attention by adding the unknown column indicator to the fields that are new or that DDF Builder changed to an unknown type.

Tip... For more information on the attributes in the grid data view, refer to [The following table lists the attributes displayed in the grid data view.](#)

A list of the issues DDF Builder detected and changed display in the Definition Errors view.



The original table definition, before DDF Builder made any changes, is available from the Original Definition view.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Currency	CURRENCY	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	7.3091420841060...
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	2	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Understanding the Errors

The Definition Errors view gives the following information:

- Where the problems are located
- What problems DDF Builder found
- What changes, if any, DDF Builder made to the original table definition

The Definition Errors lists three problems:

Where the problem is located...	The problem DDF Builder found...	What needs to be done now...
Column: unnamed_6	DDFBLDR-6: Undefined bytes in table definition. Unknown column added.	You must define the Unknown column that DDF Builder added to account for the undefined bytes.
Column: Age	DDFBLDR-11: SQL column type is not compatible with the column size. Column type changed to Unknown.	You must define the column type that DDF Builder changed to Unknown.
Column: dob	DDFBLDR-11: SQL column type is not compatible with the column size. Column type changed to Unknown.	You must define the column type that DDF Builder changed to Unknown.

The first error shows that the file contains a group of bytes that were undefined in the original table definition. DDF Builder created a new column from these undefined bytes and named the column `unnamed_6`.

The file also contains two fields (Age and dob) where the column data type was invalid for the column size. In both cases, the type for these fields has been changed to Unknown. Before you look at the Age column in depth, first review what DDF Builder can and cannot determine.

Invalid Type and Size

DDF Builder cannot confirm the size of a field based on the specified data type, but can only verify that the types available are valid for the defined size. DDF Builder can detect the problems found, make generic changes, and record the changes made, but it cannot determine if the type selected is correct.

DDF Builder limits the data types from which you can select, based on the field size. So when looking at the first invalid type and size in the Age column, for example, the size is four bytes. The Btrieve Type list in the grid data view is only populated with data types that can have a four byte size. Instead of trying each of the data types DDF Builder lets you select from, look at the data in the Btrieve Types view and the Preview page to determine an appropriate data type.

Next review the data types that can allow four bytes and preview the data in the file to see which data type is the most suitable.

Review Data Types and Sizes

To get a close look at the data with the possible data types for the field size, use the Btrieve Types view on the left side of the Table Definition Editor.

1. With the Table Definition Editor open, select the Age column so that the row is highlighted.

Tip... Selecting the Age column in the table definition allows you to see specific data in the Btrieve Types tab, as discussed next.

2. Now, click the Btrieve Types tab on the left side of the DDF Builder interface. The Btrieve Types tab view should look similar to the following:

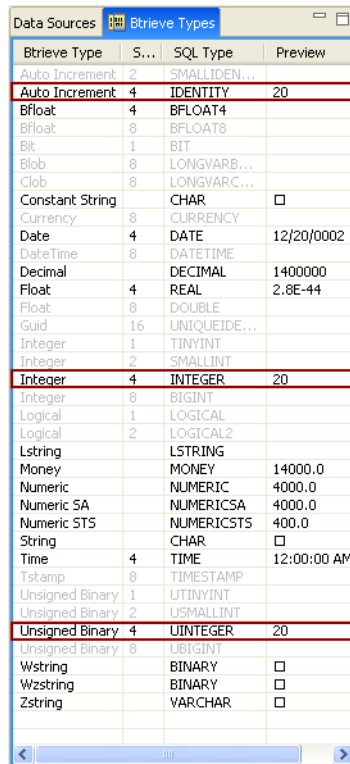
Btrieve Type	S...	SQL Type	Preview
Auto Increment	2	SMALLIDEN...	
Auto Increment	4	IDENTITY	20
Bfloat	4	BFLOAT4	
Bfloat	8	BFLOAT8	
Bit	1	BIT	
Blob	8	LONGVARB...	
Clob	8	LONGVARC...	
Constant String		CHAR	<input type="checkbox"/>
Currency	8	CURRENCY	
Date	4	DATE	12/20/0002
DateTime	8	DATETIME	
Decimal		DECIMAL	1400000
Float	4	REAL	2.8E-44
Float	8	DOUBLE	
Guid	16	UNIQUEIDE...	
Integer	1	TINYINT	
Integer	2	SMALLINT	
Integer	4	INTEGER	20
Integer	8	BIGINT	
Logical	1	LOGICAL	
Logical	2	LOGICAL2	
Lstring		LSTRING	
Money		MONEY	14000.0
Numeric		NUMERIC	4000.0
Numeric SA		NUMERICSA	4000.0
Numeric STS		NUMERICSTS	400.0
String		CHAR	<input type="checkbox"/>
Time	4	TIME	12:00:00 AM
Tstamp	8	TIMESTAMP	
Unsigned Binary	1	UTINYINT	
Unsigned Binary	2	USMALLINT	
Unsigned Binary	4	UINTEGER	20
Unsigned Binary	8	UBIGINT	
Wstring		BINARY	<input type="checkbox"/>
Wzstring		BINARY	<input type="checkbox"/>
Zstring		VARCHAR	<input type="checkbox"/>

Tip... For more information about the Btrieve Types view, see [Btrieve Types](#).

- Carefully review the data as it appears in the Preview column of the Btrieve Types view.

The Btrieve Types view only shows data types that can be four bytes in size. This is extremely helpful because it allows you to filter out all other data types that are invalid.

Knowing that the column represents an age and is four bytes in size, you can see that only three possible data types can work with the data.



Next, you can eliminate Auto Increment as a possible data type, since you know that using the Auto Increment data type automatically increases each new record by a value of one.

By a process of elimination you are now down to two possible data types from the Btrieve Types list – Integer and Unsigned Binary.

The Integer and Unsigned Binary data types are very similar in nature. Try each of these data types and see how the data gets interpreted. First, select Integer as the Btrieve data type for the Age column.

4. Select **Integer** from the Btrieve Type list.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
LastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	2	Unknown		0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
unnamed_6	53	2	Unknown		0	0	<input type="checkbox"/>	<input type="checkbox"/>	
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

You can see in the grid data view's Preview column, that the data looks acceptable. Take a moment though, and look at all the data in the file via the Preview page.

5. Click the **Preview** page in the Table Definition Editor. You can see all of the data in the file in a readable layout.

ID	Age	firstName	lastName	dob	__field_6	address	income
1	20	Joe	Smith			Austin	1000.0
2	30	James	Ogelvie			Round Rock	2000.0
3	40	Haniza	Yaacob			San Antonio	3000.0
4	50	Harold	Dimbat			Dallas	4000.0
5	60	Bradley	Giddy			Houston	5000.0

Looking at all the data in the file helps to reconcile the invalid data type with more confidence, since you can quickly look at all the records in the file.

You can see that using the Integer data type, the data is formatted in what appears to be a suitable manner. Be sure though, and run the same review using the Unsigned Binary data type.

6. Select **Unsigned Binary** from the Btrieve Type list.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Unsigned B...	UINTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	2	Unknown		0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
unnamed_6	53	2	Unknown		0	0	<input type="checkbox"/>	<input type="checkbox"/>	
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Again, the grid data view's Preview column shows the data in an acceptable format. Be certain though, and look at all the data in the file via the Preview page, as you did when you used the Integer data type.

7. Click the **Preview** page in the Table Definition Editor to see all of the data in the file in a readable layout.

ID	Age	firstName	LastName	dob	__field_6	address	income
1	20	Joe	Smith			Austin	1000.0
2	30	James	Ogelvie			Round Rock	2000.0
3	40	Hariza	Yaacob			San Antonio	3000.0
4	50	Harold	Dimbat			Dallas	4000.0
5	60	Bradley	Giddy			Houston	5000.0

Both the Integer and the Unsigned Binary data types format the data in a manner that is acceptable and understandable, and both these data types support 1, 2, 4, and 8 bytes. So it seems there is no compelling reason to select one data type over another. This example clearly shows that you must know the basic and underlying structure of your data if you are creating or modifying table definitions.

Make the Final Changes

Now that you have reviewed the possible data types and the formatted data, select the Integer data type for the Age column.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
LastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	2	Unknown		0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
unnamed_6	53	2	Unknown		0	0	<input type="checkbox"/>	<input type="checkbox"/>	
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

This resolves the first table definition error. You cannot save the table definition though, until you resolve the other errors.

Tip... You cannot save your table definitions until you resolve all Unknown types and account for every byte in the record.

The next two fields are side by side, and you should look at them collectively. To resolve the two remaining table definition errors, you need to consider them as a single error.

First, you know that DDF Builder noted that the dob field had a defined data type (Date) that is invalid for the column's size of two bytes. As a result, DDF Builder changed the data type to Unknown.

Secondly, there were two bytes that were undefined in the original table definition. DDF Builder created a new column (unnamed_6) from these two bytes and gave them an Unknown type.

The intended data type for the dob field is Date, and the Date data type requires a field of four bytes. By merging both the dob and the unnamed_6 columns, you can create a four-byte field that is appropriate for the Date data type. You need to verify though, as you did with the previous error, that the data is understandable and acceptable using the Date data type and with the two columns merged.

1. In the Table Definition Editor, select both the dob column and the unnamed_6 column (use the Shift key).
2. Right-click and select **Merge Columns**.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER		0	0	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER		0	0	<input checked="" type="checkbox"/>	20
firstName	10	20	String	CHAR(20)		0	0	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)		0	0	<input checked="" type="checkbox"/>	Smith
dob	51	2	Unknown			0	0	<input checked="" type="checkbox"/>	
unnamed_6	53	2	Unknown			0	0	<input type="checkbox"/>	
address				CHAR(50)		0	0	<input checked="" type="checkbox"/>	Austin ...
income				MONEY(8,2)		8	2	<input checked="" type="checkbox"/>	1000.0

The two columns are merged into one, making a four-byte column.

3. Select **Date** from the Btrieve Type list.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER		0	0	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER		0	0	<input checked="" type="checkbox"/>	20
firstName	10	20	String	CHAR(20)		0	0	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)		0	0	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE		0	0	<input checked="" type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)		0	0	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)		8	2	<input checked="" type="checkbox"/>	1000.0

Once you enter this information and tab through the rest of the column fields, notice that adding the Date data type formatted the data appropriately in the visible record.

It is a good idea to check the Preview page to see how all the data is handled with the Date data type.

4. Click the **Preview** tab at the bottom of the grid data view.

ID	Age	firstName	LastName	dob	address	income
1	20	Joe	Smith	09/09/1974	Austin	1000.0
2	30	James	Ogelvie	09/29/1977	Round Rock	2000.0
3	40	Hariza	Yaacob	11/18/1961	San Antonio	3000.0
4	50	Harold	Dimbat	10/12/1974	Dallas	4000.0
5	60	Bradley	Giddy	03/19/1958	Houston	5000.0

Looking at the dob column in the Preview page, you can see that all of the data is formatted in an acceptable and understandable format.

Save the Table Definition

Now that you have completed the table definition, you must save your work for the changes to take effect. Before you save the work, take one final look at the table definitions. Your data grid view should look similar to the following:

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
LastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

You can take a quick glance and see that all of the Btrieve Types have been defined, you have no undefined fields remaining and every byte is accounted for. You can now save your table definition.

1. From the menu bar, click **File > Save**.

Conclusion

This lesson introduced you to how DDF Builder handles data types and sizes that are not compatible with each other. It showed you how invalid data types and sizes might be displayed within an existing table definition and provided you with some sample solutions for modifying your table definitions so that the sizes and data types are compatible and information is proper and accurate.

Lesson 4 – Overlapping Column Definitions

Scenario

In this lesson you have an existing table definition that contains two column definitions that share some of the same bytes in the file. This creates an overlapping column definition that must be corrected.

Goals

The goal is to open the file with DDF Builder and see if DDF Builder makes any updates to the definitions. You will inspect any changes that DDF Builder implements and discuss changes needed to fix the overlapping column definition.

Note: DDF Builder recommends solutions to fix the overlapping column definitions in your file and allows you to save those changes.

What You Need to Know

For this lesson, use the file named OVERLAP.MKD. This file resides in a folder named Tutorial2. Assuming you installed using the default installation locations, this folder is located at:

```
<Application Data>\DDFBuilder\tutorials\tutorial2
```

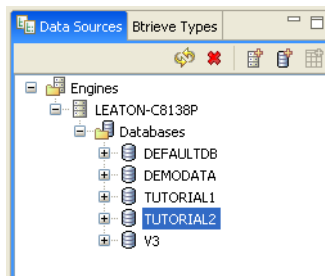
This folder is part of the Tutorial2 database.

Note: You must have a Data Source Name (DSN) that points to this database in order to access the data in this tutorial. If you have not yet created this DSN, refer to [Create Data Source Names \(DSN\)](#).

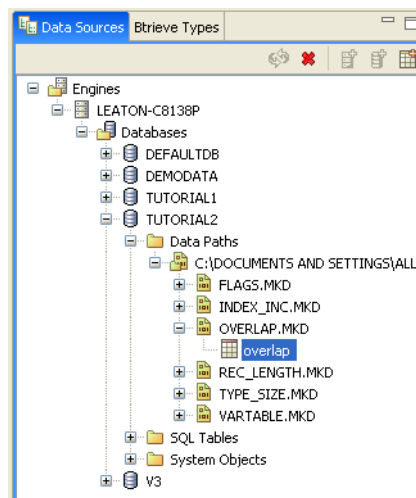
Open the Btrieve File

You should have DDF Builder already running from the last lesson. If not, begin by starting DDF Builder.

1. In the DDF Builder expand the Data Sources Explorer nodes, and locate the Tutorial2 database in the list.

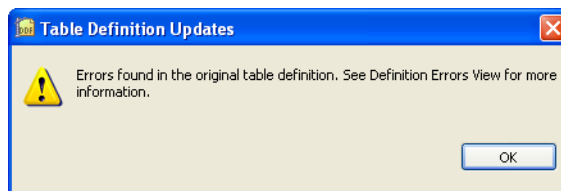


2. Double-click the Tutorial2 database icon to expand the nodes.
3. Expand the nodes until the OVERLAP.MKD file and the associated SQL table display.



4. Double-click the overlap SQL table, or right-click and select **Edit Table Definition**.

The Table Definition Editor opens and displays the following message:



This message tells you that DDF Builder has analyzed your existing table definitions and found problems with those definitions. As a result of this, DDF Builder had to make some modifications to open and display the existing table definitions.

5. Click **OK** to clear the message and display the table definition.

Tip... For a complete list of possible definition errors, refer to [Definition Errors List](#).

Look for Inconsistencies

Begin by looking at the DDF Builder interface to see the differences between the original definition and the modifications that DDF Builder made, as well as reviewing the errors reported.

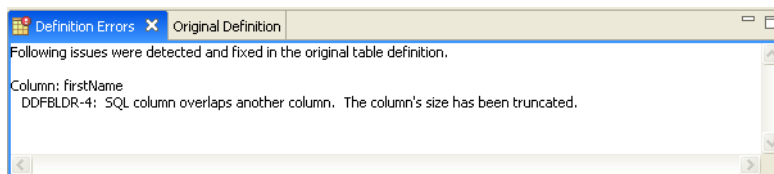
Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith ...
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Table | Indexes | Preview | Statistics | SQL View

The Table Definition Editor's grid data view shows the existing table definition with the modifications made by DDF Builder.

No unknown icons or other visual indicators bring to your attention the fields that are altered or added.

The Definition Errors window displays the issues DDF Builder detected and changed.



The original table definition, before DDF Builder made any changes, is always available from the Original Definition view.

Note: DDF Builder changes are not automatically saved. Any modifications made by DDF Builder must be saved.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	30	String	CHAR(30)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe Smith
LastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

In a moment compare the original definition to the modified definition, but first take a look at the error DDF Builder reported.

Understanding the Errors

The Definition Errors view tells the following:

- where the problems are located
- what problems DDF Builder found
- what changes, if any, DDF Builder made to the original table definition

The Definition Errors lists one problem; look at that problem a bit more in-depth.

Where the problem is located...	The problem DDF Builder found...	What needs to be done now...
Column: firstName	DDFBLDR-4: SQL column overlaps another column. The column's size has been truncated.	The column that DDF Builder named (firstName) overlaps another column. To eliminate the overlapping columns and make certain that both the table definition and the Btrieve file have the same record length, DDF Builder shortened the column size.

The error DDF Builder reported is an overlapping column definition. This means the table definition contains definitions for two columns that share some of the same bytes.

DDF Builder automatically reduced the size of the firstName column so that it no longer overlaps with the next column. Now the column lengths match and no bytes are defined across multiple columns.

You can see the overlapping columns by comparing the grid data view with the original definition.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	30	String	CHAR(30)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe Smith
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Notice in the original definition the firstName field has a size of 30, compared to a size of 20 in the modified definition. The original definition shows both a first name (Joe) and last name (Smith) in the Preview column. The modified definition, with the smaller size, only shows the first name in the Preview column, as you would expect.

Accept or Reject Changes

Making comparisons between the original definition and the definition that DDF Builder modifies, along with verifying how the data is formatted, helps you determine whether or not to accept the changes made by DDF Builder.

As a result of the changes that DDF Builder made, the data appears more in a manner that you would expect, and you can be assured that column lengths match.

Now that you have reviewed the differences in the files, understand the changes that DDF Builder made, and can verify the data is proper, you are ready to accept the changes.

Save the Table Definition

Any time that changes are made to your table definitions, whether they are made by you or by DDF Builder, they must be saved in order for them to take effect. Before you save the table definition, it is a good idea to take one final look at the definition and the Preview tab to make certain that all fields are defined and every byte is accounted for.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith ...
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Table | Indexes | Preview | Statistics | SQL View

Note: You cannot save a table definition with fields that contain a Btrieve Type of Unknown or that contain bytes that are unaccounted for.

You can see all of the Btrieve Types are defined and every byte in the record is accounted for. You can now save your table definition.

1. From the menu bar, click **File > Save** to save the table definition.

The table definition DDF Builder modified to fix the overlapping column definition is now saved as your current definition.

Conclusion

This lesson introduced you to how DDF Builder deals with overlapping column definitions and the kind of changes DDF Builder implements to resolve overlapping columns. It showed you ways to compare the definition modified by DDF Builder with the original definition to determine if changes should be accepted or rejected.

Lesson 5 – File/Field Flag Inconsistencies

Scenario

In this lesson, you have a Btrieve file with flags set for a field that are inconsistent with the SQL column flags set in the table definition.

Goals

The goal is to open the file with DDF Builder and see if DDF Builder makes any updates to the definitions. You will inspect any changes that DDF Builder implements and discuss changes, if any, that are needed to fix the flag inconsistencies.

What You Need to Know

For this lesson, used the file named FLAGS.MKD. This file resides in a folder named Tutorial2. Assuming you installed using the default installation locations, this folder is located at:

```
<Application Data>\DDFBuilder\tutorials\tutorial2
```

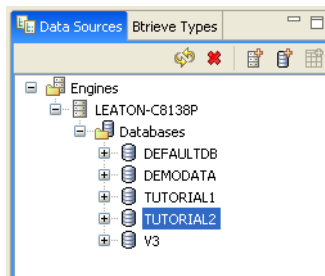
This folder is part of the Tutorial2 database.

Note: You must have a Data Source Name (DSN) that points to this database in order to access the data in this tutorial. If you have not yet created this DSN, refer to [Create Data Source Names \(DSN\)](#).

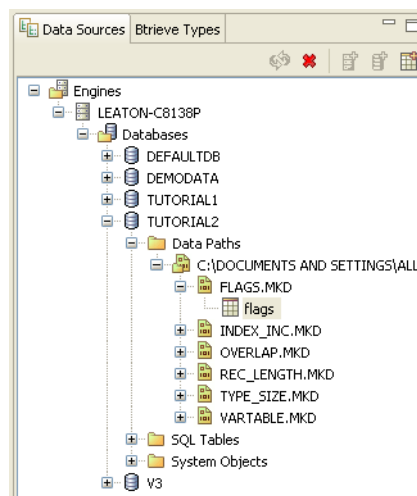
Open the Btrieve File

You should have DDF Builder already running from the last lesson. If not, begin by starting DDF Builder.

1. In the DDF Builder expand the Data Sources Explorer nodes, and locate the Tutorial2 database in the list.

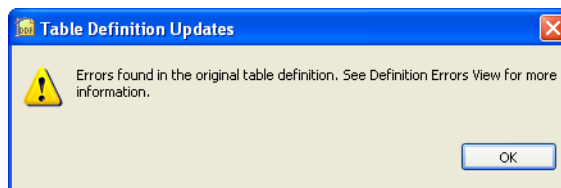


2. Double-click the Tutorial2 database icon to expand the nodes.
3. Expand the nodes until the FLAGS.MKD file and the associated SQL table display.



4. Double-click the flags SQL table, or right-click and select **Edit Table Definition**.

The Table Definition Editor opens and displays the following message:



This message tells you that DDF Builder analyzed your existing table definitions and found problems with those definitions. As a result of this, DDF Builder had to make some modifications to open and display the existing table definitions.

5. Click **OK** to clear the message and display the table definition.

Tip... For a complete list of possible definition errors, refer to [Definition Errors List](#).

Look for Inconsistencies

Begin by looking at the DDF Builder interface to review the inconsistencies with which you are dealing. First look at the grid data view and the Definition Errors window.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	CURRENCY	0	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Table | Indexes | Preview | Statistics | SQL View

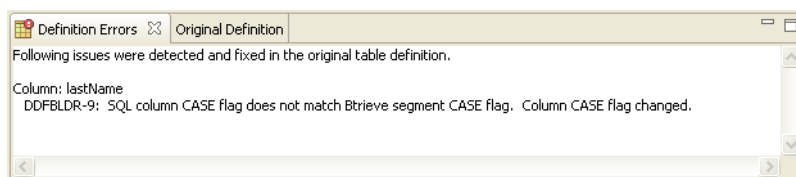
The grid data view of the Table Definition Editor shows your existing table definition with the modifications that DDF Builder made.

Note: DDF Builder changes are not automatically saved. Any modifications made by DDF Builder must be saved.

For this lesson, DDF Builder does not give any visual indications as to the columns that need attention. No unknown column indicators suggest that any of the fields have been altered.

Tip... For more information on the attributes in the grid data view, refer to [The following table lists the attributes displayed in the grid data view.](#)

A list of the issues DDF Builder detected and changed display in the Definition Errors view.



Now, take a moment to look at the original table definition. The original table definition, before DDF Builder made any changes, is available from the Original Definition view.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	CURRENCY	0	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Before you proceed with the details of the inconsistencies in the table definitions, make sure that you understand the definition errors reported by DDF Builder.

Understanding the Errors

The Definition Errors view tells the following:

- where the problems are located
- what problems DDF Builder found
- what changes, if any, DDF Builder made to the original table definition

The Definition Errors list one problem:

Where the problem is located...	The problem DDF Builder found...	What needs to be done now...
Column: lastName	DDFBLDR-9: SQL column CASE flag does not match Btrieve segment CASE flag. Column CASE flag changed.	You need to verify if the field should have the CASE flag set or not and either accept or reject the DDF Builder changes.

The error indicates a Case flag set on a SQL column but the flag was not set on the corresponding Btrieve segment. Since DDF Builder does not change the Btrieve file, the SQL table has been modified to match the specifics of the Btrieve file. To make certain this is the situation, compare the current table definition to the original definition.

The definition error lists the problem is with the lastName column's Case setting. If you compare the lastName fields of the current and original table definition, you can quickly see the difference.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

From the comparison, you can determine the following:

- The original, or existing, definition did *not* have the Case flag set on the lastName field.
- The current definition (updated by DDF Builder) *does* have the Case flag set on the lastName field.

DDF Builder made changes based upon the Btrieve file. Remember, DDF Builder cannot change your Btrieve file, only the table definitions that determine the structure of your Btrieve file. That being the case, DDF Builder changed the original definition (without the Case flag) to allow for the lastName field to use the Case flag.

Note: The lastName field is used as an index. You cannot clear the **Case** flag from the lastName field because it is an index.

Accept or Reject Changes

You have determined the cause of the definition error and understand what changes DDF Builder made. Now you need to either accept or reject the changes that DDF Builder made to the table definition.

If the changes DDF Builder made are incorrect, you would reject the changes and create a new table definition from scratch, since you are unable to change the Case setting for a field marked as an index.

However, for the purpose of this lesson, accept the changes that DDF Builder made. You want the Case flag set on the lastName field.

Save the Table Definition

Now that you have reviewed the table definition error and changes, you must save your work in order for the change to take effect. Before you save your work, take one final look at the table definition. Your data grid view should look similar to the following:

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	CURRENCY	0	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

Table | Indexes | Preview | Statistics | SQL View

Take a quick glance and verify that all of the Btrieve Types are defined, there are no undefined fields remaining and every byte is accounted for. You can now save your table definition.

1. From the menu bar, click **File > Save** to save the table definition.

Your updated table definition is now saved to use the Case flag setting on the lastName field.

Conclusion

This lesson introduced you to how DDF Builder handles flag settings that differ from the Btrieve file and the existing table definition. It showed you how the grid data view and the original definitions view can be used to compare changes made by DDF Builder. This lesson also showed you that altering a field set as an index is not allowed in DDF Builder. Strategies for rejecting or accepting the changes made by DDF Builder were also discussed.

Lesson 6 – Index Inconsistencies

Scenario

In this lesson, you have a Btrieve file with indexes set that are inconsistent with the indexes set in the table definition.

Goals

The goal is to open the file with DDF Builder and see if DDF Builder updates any of the definitions. You will inspect any changes that DDF Builder implements and discuss changes that are needed to fix the index inconsistencies.

Tip... DDF Builder recommends solutions to fix the inconsistencies in your indexes and allows you to save those changes.

What You Need to Know

For this lesson, use the file named INDEX_INC.MKD. This file resides in a folder named Tutorial2. Assuming you installed using the default installation locations, this folder is located at:

```
<Application Data>\DDFBuilder\tutorials\tutorial2
```

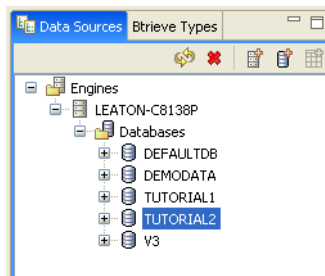
This folder is part of the Tutorial2 database.

Note: You must have a Data Source Name (DSN) that points to this database in order to access the data in this tutorial. If you have not yet created this DSN, refer to [Create Data Source Names \(DSN\)](#).

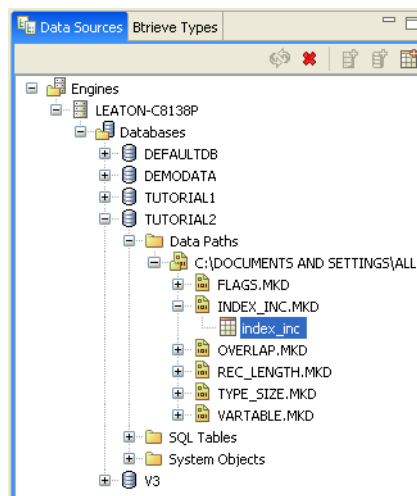
Open the Btrieve File

You should have DDF Builder already running from the last lesson. If not, begin by starting DDF Builder.

1. In the DDF Builder expand the Data Sources Explorer nodes, and locate the Tutorial2 database in the list.

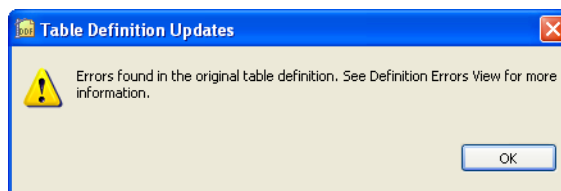


2. Double-click the Tutorial2 database icon to expand the nodes.
3. Expand the nodes until all the files display.



4. Double-click the index_inc SQL table, or right-click and select **Edit Table Definition**.

The Table Definition Editor opens and displays the following message:



This message tells you that DDF Builder has analyzed your existing table definitions and found problems with those definitions. As a result of this, DDF Builder had to make some modifications to open and display the existing table definitions.

5. Click **OK** to clear the message and display the table definition.

Tip... For a complete list of possible definition errors, refer to [Definition Errors List](#).

Look for Inconsistencies

Begin by looking at the DDF Builder interface to review the inconsistencies found. Start by doing a quick comparison of the grid data view to the original definition.

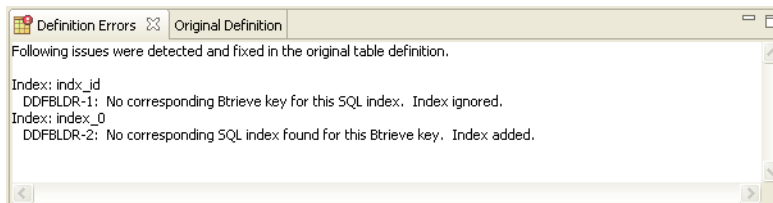
The screenshot shows the DDF Builder interface. At the top is a table definition grid with columns: Field, Offset, Size, Btrieve Type, SQL Type, Precision, Scale, Null, Case, and Preview. Below the grid is a toolbar with buttons for Table, Indexes, Preview, Statistics, and SQL View. Below the toolbar is a window titled 'Definition Errors' with a tab labeled 'Original Definition'. This window contains a table definition grid identical to the one above.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

For this lesson, DDF Builder does not give any visual indications as to the columns that need attention. No unknown column indicators suggest that any of the fields have been altered.

Interestingly enough, you can compare these two definitions and find no differences. Although DDF Builder has made changes, they do not appear in the table definition views because the issues DDF Builder corrected are in the indexes.

Before you look at the index information in detail, take a moment to review the problems listed in the Definition Errors window. A list of the issues DDF Builder detected and changed display in the Definition Errors view.



Note: DDF Builder changes are not automatically saved. Any modifications made by DDF Builder must be saved.

Understanding the Errors

The Definition Errors view tells the following:

- where the problems are located
- what problems DDF Builder found
- what changes, if any, DDF Builder made to the original table definition

Tip... DDF Builder cannot alter your Btrieve file. Any changes made are to the table definition and not the Btrieve file.

The Definition Errors lists two problems:

Where the problem is located...	The problem DDF Builder found...	What needs to be done now...
Index: indx_id	DDFBLDR-1: No corresponding Btrieve key for this SQL index. Index ignored.	A corresponding key does not exist in the Btrieve file for a SQL index found in the table definition. Therefore, DDF Builder ignored the index, and the only action is to confirm the changes made by DDF Builder.
Index: index_0	DDFBLDR-2: No corresponding SQL index found for this Btrieve key. Index added.	The Btrieve file has the key and DDF Builder has added the corresponding index in the table definition. You need to verify the index in the table definition and accept the changes DDF Builder made.

The first error shows that the existing table definition contains an index that does not exist in the Btrieve file. Because DDF Builder cannot change the Btrieve file – only alter table definitions – the index is ignored. This is important to know so that you do not try and use an index that does not exist.

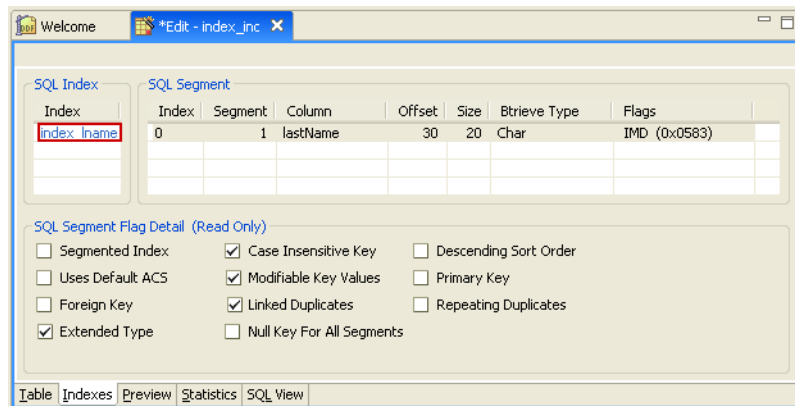
The second error is the opposite situation. The existing table definition does not contain an index that is defined in the Btrieve file. DDF Builder added a SQL index to represent the index in the Btrieve file that was previously undefined.

Assign a name to the index that DDF Builder created, and accept the changes made to the table definition index.

Note: In the event you have indexes created by DDF Builder that are no longer needed, you may benefit from dropping the index from the file and the table definition. Similarly, you may find that you need to add indexes that were not defined when DDF Builder modified your table definitions. In either case, refer to the *Zen Programmer's Guide* for information on adding and dropping indexes.

Name the Index

1. Select the **Indexes** tab in the Table Definition Editor.
2. Double-click the `index_0` entry in the Index column.
3. Enter `index_lname` as the name of the index.



Note: To see the indexes defined on the Btrieve file, click the Statistics tab.

Save the Table Definition

Now that you verified the index DDF Builder added to the table definition, save your work for the changes to take effect. Before you save your work, take one final look at the table definition.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
address	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Austin ...
income	107	5	Money	MONEY(8,2)	8	2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1000.0

All of the Btrieve Types have been defined, there are no undefined fields remaining and every byte is accounted for. You can now save your table definition.

1. From the menu bar, click **File** > **Save** to save the table definition.

Conclusion

Congratulations, you have successfully saved your table definition. This lesson introduced you to some of the inconsistencies in indexes that DDF Builder recognizes. It also covered the modifications that DDF Builder makes and how you save the updated table definition.

Lesson 7 – Variable Length Record Mismatch

Scenario

In this lesson, you have a Btrieve file with a different variable length record than the variable length record set in the table definition.

Goals

The goal is to open the file with DDF Builder and see if DDF Builder makes any updates to the definitions. You will inspect any changes that DDF Builder implements and discuss changes needed to make to fix the variable length record mismatch.

Note: Currently DDF Builder does not recommend solutions to fix the variable length record mismatch.

What You Need to Know

For this lesson, use the file named VARIABLE.MKD. This file resides in a folder named Tutorial2. Assuming you installed using the default installation locations, this folder is located at:

```
<Application Data>\DDFBuilder\tutorials\tutorial2
```

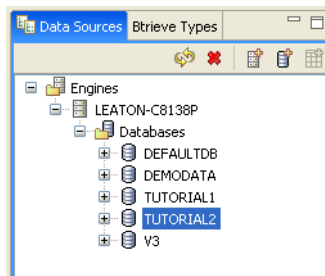
This folder is part of the Tutorial2 database.

Note: You must have a Data Source Name (DSN) that points to this database in order to access the data in this tutorial. If you have not yet created this DSN, refer to [Create Data Source Names \(DSN\)](#).

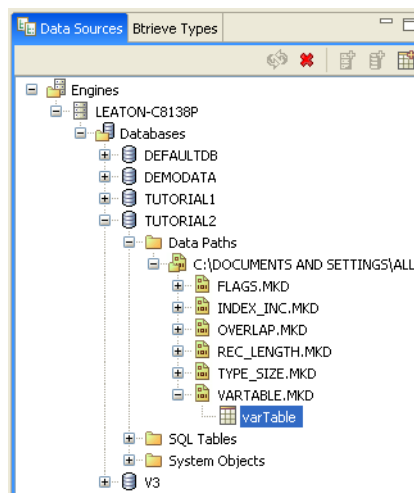
Open the Btrieve File

You should have DDF Builder already running from the last lesson. If not, begin by starting DDF Builder.

1. In the DDF Builder expand the Data Sources Explorer nodes, and locate the Tutorial2 database in the list.

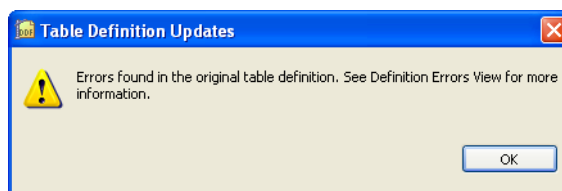


2. Double-click the Tutorial2 database icon to expand the nodes.
3. Expand the nodes until all the files display.



4. Double-click the var table SQL table, or right-click and select **Edit Table Definition**.

The Table Definition Editor opens and displays the following message:



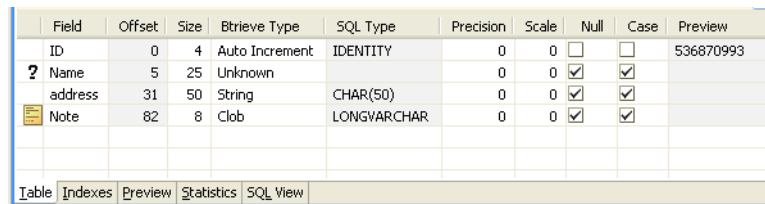
This message tells you that DDF Builder has analyzed your existing table definitions and found problems with those definitions. As a result of this, DDF Builder had to make some modifications to open and display the existing table definitions.

5. Click **OK** to clear the message and display the table definition.

Tip... For a complete list of possible definition errors, refer to [Definition Errors List](#).

Look for Inconsistencies

Begin by looking at the DDF Builder interface to review the differences between the original definition and the modifications that DDF Builder made, as well as reviewing the errors reported.



Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Auto Increment	IDENTITY	0	0	<input type="checkbox"/>	<input type="checkbox"/>	536870993
? Name	5	25	Unknown		0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
address	31	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Note	82	8	Clob	LONGVARCHAR	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

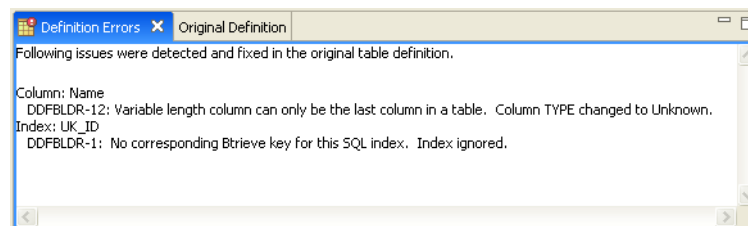
The Table Definition Editor's grid data view shows the modified table definition.

Note: DDF Builder changes are not automatically saved. Any modifications made by DDF Builder must be saved.

DDF Builder gives you visual indications as to the columns that need attention by adding the unknown column indicator to the fields that are altered or added. In addition, the variable column indicator is added to columns that contain an unknown variable.

Tip... For more information on the attributes in the grid data view, refer to [The following table lists the attributes displayed in the grid data view.](#)

A list of the issues DDF Builder detected and changed display in the Definition Errors view.



The original table definition, before DDF Builder made any changes, is available from the Original Definition view.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Auto Increment	IDENTITY	0	0	<input type="checkbox"/>	<input type="checkbox"/>	536870993
Name	5	25	Note	NOTE(25)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
address	31	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Note	82	8	Clob	LONGVARCHAR	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Understanding the Errors

The Definition Errors view tells the following:

- where the problems are located
- what problems DDF Builder found
- what changes, if any, DDF Builder made to the original table definition.

The Definition Errors lists two problems:

Where the problem is located...	The problem DDF Builder found...	What needs to be done now...
Column: Name	DDFBLDR-12: Legacy variable length column can only be the last column in a table. Column TYPE changed to Unknown.	The existing SQL table definition contains a NOTE or LVAR data type column in the middle of the table. DDF Builder changed the data type to Unknown because variable length columns must be the last column in the table. You must define the column that DDF Builder changed to Unknown.
Index: UK_ID	DDFBLDR-1: No corresponding Btrieve key for this SQL index. Index ignored.	A corresponding key does not exist in the Btrieve file for a SQL index found in the table definition. Therefore, DDF Builder ignored the index, and the only action is to confirm the changes made by DDF Builder.

The DDFBLDR-12 error results from a variable length field that does not occur as the last column in the table definition. Because the file contained a variable length field in a place other than the last column, DDF Builder changed the data type of this field to Unknown.

Tip... [Lesson 3 – Invalid Data Types and Sizes](#) provides a step-by-step tutorial specifically dealing with the situation where the column data type is invalid for the column size.

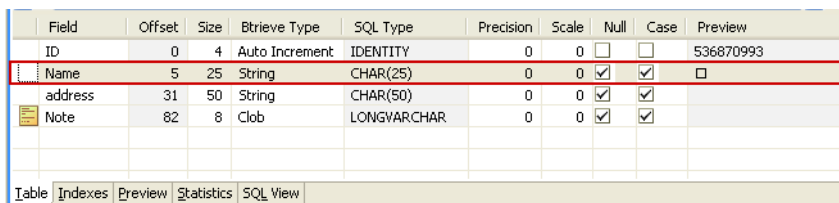
To fix the variable length field issue, define the unknown field with an appropriate data type, as discussed in the next section.

The last error reported is an index that DDF Builder found in the existing table definition that does not have a corresponding key in the Btrieve file. In the course of this lesson, this error is not addressed, only the variable length field. For information on resolving errors pertaining to indexes, see [Lesson 6 – Index Inconsistencies](#).

Define the Unknown Field

Define the unknown field in the grid data view of the Table Definition Editor.

1. Click to select the **Name** field.
2. Select **String** from the Btrieve Type list.



Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Auto Increment	IDENTITY	0	0	<input type="checkbox"/>	<input type="checkbox"/>	536870993
Name	5	25	String	CHAR(25)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
address	31	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Note	82	8	Clob	LONGVARCHAR	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

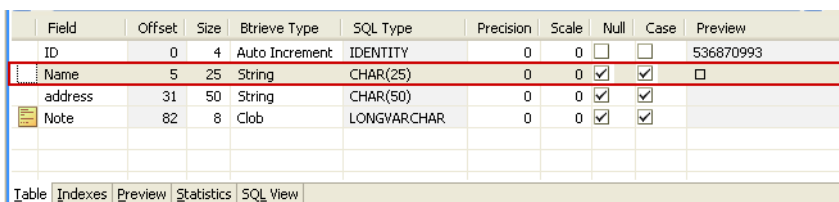
Table | Indexes | Preview | Statistics | SQL View

Note: Editing columns in a variable portion of a table is not recommended if your definition contains BLOB or CLOB data types.

Save the Table Definition

You have defined the Unknown field and fixed all of the issues, now save the table definition.

Before you save your work, take one final look at the table definition.



Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Auto Increment	IDENTITY	0	0	<input type="checkbox"/>	<input type="checkbox"/>	536870993
Name	5	25	String	CHAR(25)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
address	31	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Note	82	8	Clob	LONGVARCHAR	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Table | Indexes | Preview | Statistics | SQL View

The table definition shows no undefined fields remaining and every byte is accounted for. You can now save your table definition.

-
1. From the menu bar, click **File** > **Save** to save the table definition.

Conclusion

Congratulations, you have successfully saved your table definition and corrected your variable length field. This lesson introduced you to how DDF Builder deals with variable length fields that are not at the end of the record. It also showed you how to define the field and create a valid table definition for the file.

Lesson 8 – Record Length Mismatch

Scenario

In this lesson, you have a Btrieve file with a different record length than the record length set in the table definition.

Goals

The goal is to open the file with DDF Builder and see if DDF Builder makes any updates to the definitions. You will inspect any changes that DDF Builder implements and discuss changes that are needed to fix the record length mismatch.

Tip... DDF Builder recommends solutions to fix the record length mismatch and allows you to save those changes.

What You Need to Know

For this lesson, use the file named `rec_length.mkd`. This file resides in a folder named `Tutorial2`. Assuming you installed using the default installation locations, this folder is located at:

```
<Application Data>\DDFBuilder\tutorials\tutorial2
```

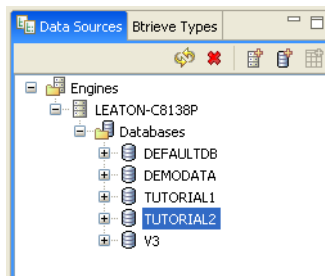
This folder is part of the `Tutorial2` database.

Note: You must have a Data Source Name (DSN) that points to this database in order to access the data in this tutorial. If you have not yet created this DSN, refer to [Create Data Source Names \(DSN\)](#).

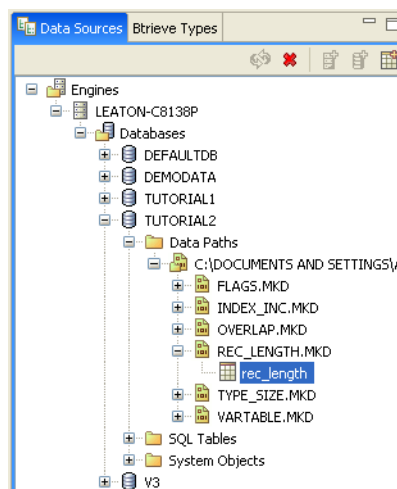
Open the Btrieve File

You should have DDF Builder already running from the last lesson. If not, begin by starting DDF Builder.

1. In the DDF Builder expand the Data Sources Explorer nodes, and locate the `Tutorial2` database in the list.

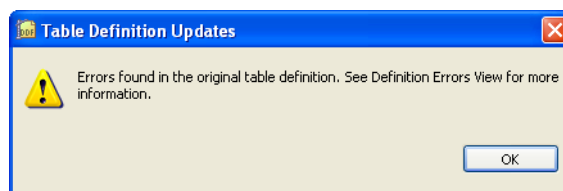


2. Double-click the Tutorial2 database icon to expand the nodes.
3. Expand the nodes until all the files display.



4. Double-click the rec_length SQL table, or right-click and select **Edit Table Definition**.

The Table Definition Editor opens and displays the following message:



This message tells you that DDF Builder has analyzed the existing table definitions and found problems with those definitions. As a result of this, DDF Builder had to make some modifications to open and display the existing table definitions.

5. Click **OK** to clear the message and display the table definition.

Tip... For a complete list of possible definition errors, refer to [Definition Errors List](#).

Look for Inconsistencies

Begin by looking at the DDF Builder interface to review the differences between the original definition and the modifications that DDF Builder made, as well as reviewing the errors reported.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
? unnamed_6	55	57	Unknown		0	0	<input type="checkbox"/>	<input type="checkbox"/>	

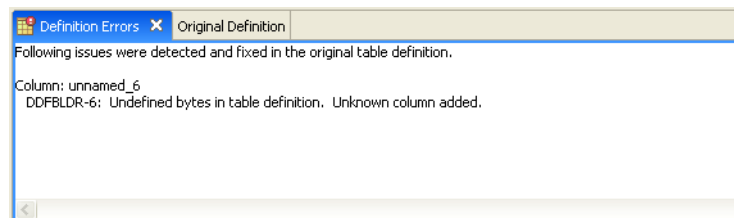
The Table Definition Editor's grid data view shows the modified table definition.

Note: DDF Builder changes are not automatically saved. Any modifications made by DDF Builder must be saved.

DDF Builder gives you visual indications as to the columns that need attention by adding the unknown column indicator to the fields that are altered or added.

Tip... For more information on the attributes in the grid data view, refer to [The following table lists the attributes displayed in the grid data view.](#)

A list of the issues DDF Builder detected and changed display in the Definition Errors view.



The original table definition, before DDF Builder made any changes, is available from the Original Definition view.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974

Understanding the Errors

The Definition Errors view tells the following:

- where the problems are located
- what problems DDF Builder found
- what changes, if any, DDF Builder made to the original table definition.

The Definition Errors lists one problem:

Where the problem is located...	The problem DDF Builder found...	What needs to be done now...
Column: unnamed_6	DDFBLDR-6: Undefined bytes in table definition. Unknown column added.	You must define the Unknown column that DDF Builder added to account for the undefined bytes.

The error reported by DDF Builder indicates a record length mismatch.

Tip... You can see the original record length for the Btrieve file by clicking the **Statistics** tab.

This means that the table definition did not contain fields to account for all the bytes in the record. DDF Builder created a new column with the unknown data type to account for these bytes.

Defining and accounting for all of the bytes creates a table definition that accurately matches the Btrieve file. Look at the bytes at the bottom of the grid data view in the row named unnamed_6.

Field	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Null	Case	Preview
ID	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	<input type="checkbox"/>	1
Age	5	4	Integer	INTEGER	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
firstName	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Joe
lastName	30	20	String	CHAR(20)	0	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Smith
dob	51	4	Date	DATE	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	09/09/1974
? unnamed_6	55	57	Unknown		0	0	<input type="checkbox"/>	<input type="checkbox"/>	

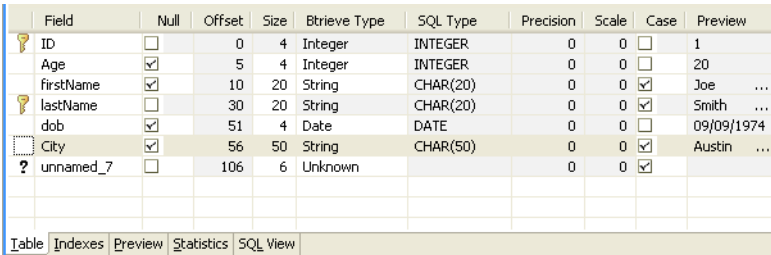
Now define the unknown field so that the record lengths match between the file and the table definition.

Define the Field

Next, define the unknown field in the grid data view of the Table Definition Editor.

1. Click to select the last row in the grid data view.
2. Select the unnamed_6 Field and enter City for the field name.
3. Check the **Null** check box.
4. Change the **Size** to reflect 50 bytes.
5. In the **Btrieve Type** column, select **String** from the list.
6. Select **Case** check box.

Your table definition should now look similar to the following:



Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
Age	<input checked="" type="checkbox"/>	5	4	Integer	INTEGER	0	0	<input type="checkbox"/>	20
firstName	<input checked="" type="checkbox"/>	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	Joe ...
lastName	<input type="checkbox"/>	30	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	Smith ...
dob	<input checked="" type="checkbox"/>	51	4	Date	DATE	0	0	<input type="checkbox"/>	09/09/1974
City	<input checked="" type="checkbox"/>	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	Austin ...
? unnamed_7	<input type="checkbox"/>	106	6	Unknown		0	0	<input checked="" type="checkbox"/>	

7. Now, click to select the unnamed_7 row in the grid data view.
8. Select the unnamed_7 Field and enter Income for the field name.
9. In the **Btrieve Type** column select Money from the list.
10. Leave the **Precision** at 10; the **Scale** should remain at 2.

Your table definition should now look similar to the following:

Field	Null	Offset	Size	Btrieve Type	SQL Type	Precision	Scale	Case	Preview
ID	<input type="checkbox"/>	0	4	Integer	INTEGER	0	0	<input type="checkbox"/>	1
Age	<input checked="" type="checkbox"/>	5	4	Integer	INTEGER	0	0	<input type="checkbox"/>	20
firstName	<input checked="" type="checkbox"/>	10	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	Joe ...
lastName	<input type="checkbox"/>	30	20	String	CHAR(20)	0	0	<input checked="" type="checkbox"/>	Smith ...
dob	<input checked="" type="checkbox"/>	51	4	Date	DATE	0	0	<input type="checkbox"/>	09/09/1974
City	<input checked="" type="checkbox"/>	56	50	String	CHAR(50)	0	0	<input checked="" type="checkbox"/>	Austin ...
Income	<input type="checkbox"/>	106	6	Money	MONEY(10,2)	10	2	<input checked="" type="checkbox"/>	1000.0

Now that all the unknown fields are defined and every byte is accounted for, save the table definition.

Save the Definition

Save the table definition so that you save your work and apply the changes.

1. From the menu bar, click **File > Save** to save the table definition.

Conclusion

Congratulations, you have successfully saved your table definition and corrected the record length mismatch. This lesson introduced you to how DDF Builder deals with differing record lengths. It also showed you how to define the field, split the column by resizing and create a valid table definition for the file.