



**Transoft U/SQL
User Guide**

Copyright © 2004-2007 Computer Software Group

This document is copyright and all rights are reserved. It may not, in whole or in part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior permission in writing from Transoft.

Transoft, part of Computer Software Group, reserves the right to make changes in specifications and other information contained in this document without prior notice.

Whilst Transoft endeavours to ensure the accuracy of the contents of this document, no liability is accepted for any error or omission.

Transoft and e-evolutionary solutions are either registered trademarks and/or trademarks of Transoft in various countries throughout the world. All other marks, names and logos may be the trademarks of their respective owners.

Transoft (Atlanta office)

1165 Northchase, Suite 375, Marietta, GA 30067, USA

Phone: +1 (770) 933 1965

Fax: +1 (770) 933 3464

Transoft (Dayton office)

7333 Paragon, Suite 250, Dayton, OH 45459, USA

Phone: +1 (937) 438 5553

Fax: +1 (937) 438 5377

Transoft (UK office)

Transoft House, 5J Langley Business Centre, Station Road, Langley, Slough, SL3 8DS, England

Phone: +44 (0) 1753 778000

Fax: +44 (0) 1753 773050

Please use e-mail to contact us wherever possible:

Americas: **USSupport@transoft.com**

Rest of the World: **UKSupport@transoft.com**

Web site:

<http://www.transoft.com>

<http://www.computersoftware.com/>

Table Of Contents

| | |
|--|----|
| Introduction..... | 1 |
| Installation and Licensing | 3 |
| Installation and Licensing | 3 |
| Installing the Multiple-Tier U/SQL Server | 4 |
| Server platforms supported..... | 4 |
| Installing the UNIX U/SQL Server Software | 4 |
| Installing the Windows NT U/SQL Server Software | 6 |
| Uninstalling U/SQL Server on Windows NT Server | 9 |
| The Next Steps..... | 9 |
| Installing the Single or Multiple-tier U/SQL client..... | 10 |
| System Requirements..... | 10 |
| Contents of U/SQL Client Software | 10 |
| Installing the Client Software | 12 |
| De-installing the U/SQL Client Software..... | 18 |
| The Next Step | 19 |
| Installing and Licensing Vista Business edition..... | 20 |
| Installing HTTP Server on Windows | 24 |
| U/SQL Client Browser on Windows | 24 |
| Getting started with the HTTP Server | 26 |
| Limitations..... | 28 |
| Installing HTTP Server on UNIX | 30 |
| U/SQL HTTP Client Browser on UNIX | 30 |
| Getting started with the HTTP Server | 31 |
| Limitations..... | 33 |
| Licensing | 35 |
| Licensing | 35 |
| Installing the U/SQL Client License..... | 36 |
| Installing the U/SQL Server License..... | 41 |
| Installing the Server License on UNIX Platforms..... | 42 |
| Installing the Server License on Windows NT Server..... | 47 |
| Starting U/SQL with Licensing | 51 |
| Unattended Client installation | 55 |
| Unattended Client installation..... | 55 |
| Introduction | 55 |
| How to perform Unattended (Silent) installations | 55 |
| InstallShield - Common error codes | 59 |
| Configure and Use | 61 |
| ODBC Overview | 61 |
| ODBC Overview | 61 |

Table Of Contents

| | |
|--|-----|
| ODBC Compliance | 65 |
| System and User Data Sources | 71 |
| File Data Sources | 72 |
| Single-tier Administration | 77 |
| Installed Files | 77 |
| U/SQL Administrator | 78 |
| Entering the ODBC.INI Directives | 79 |
| Multiple-tier Administration (Windows) | 82 |
| U/SQL Server Installation on Windows | 82 |
| U/SQL Service Manager | 84 |
| U/SQL Server Directives | 89 |
| U/SQL Administrator Facilities | 91 |
| Multiple-tier Administration (UNIX) | 95 |
| U/SQL Server Installation on UNIX | 95 |
| Starting and Stopping the UNIX Server | 98 |
| Setting up the UNIX usqlsd.ini File | 99 |
| Micro Focus COBOL Specific Issues | 103 |
| Additional Information | 104 |
| Client Configuration | 105 |
| U/SQL Client Installation Files | 105 |
| Interactive U/SQL Utilities | 106 |
| Win U/SQLi | 107 |
| usqli on UNIX Servers | 123 |
| JDBC Client | 133 |
| INI Directives | 134 |
| ODBC.INI Directives | 134 |
| UNIX Client-Side Directives | 136 |
| Configuration Section | 137 |
| Data Source Defaults Section | 140 |
| Data Source Section | 145 |
| Foreign Character Set Support | 146 |
| Advanced Directives | 147 |
| Validation Rules for Security Directives | 155 |
| COGNOS Impromptu Outer Join Directives | 157 |
| The USQLCS.LOG File and Error Reporting | 158 |
| The USQLCS.LOG File and Log Levels | 158 |
| Error Messages | 163 |
| SQL Support | 167 |
| SQL Syntax Supported | 167 |
| Sample Queries | 168 |

| | |
|--|-----|
| READ_ONLY Views | 176 |
| Limitations | 178 |
| Transaction Processing | 179 |
| Locking..... | 182 |
| Security..... | 183 |
| Security..... | 183 |
| Multiple-tier Security - User Connection | 184 |
| GRANT and REVOKE Security..... | 188 |
| Query Planning..... | 194 |
| Query Planner | 194 |
| Tuning the Query Planner | 199 |
| Query Planner Hinting | 209 |
| Sample Applications..... | 211 |
| Demonstration - Books Wholesaler..... | 211 |
| Writing Applications Using U/SQL To Access Your Data | 218 |
| General UDD Information | 223 |
| Overview | 223 |
| Creating a UDD..... | 225 |
| Expression Handling | 229 |
| Handling of Data Arrays..... | 232 |
| Limitations..... | 247 |
| Specific UDD Information..... | 248 |
| Setting Up a COBOL Data Dictionary | 249 |
| Setting Up a C-ISAM Data Dictionary | 352 |
| Setting Up a Business BASIC Data Dictionary | 370 |
| Setting Up a U/FOS Data Dictionary | 391 |
| Enterprise Join Engine..... | 411 |
| Enterprise Join Engine | 411 |
| Using the Enterprise Join Engine | 413 |
| Configuring the EJE on UNIX | 420 |
| Configuring Multi-Datasource U/SQL EJE on UNIX | 422 |
| Configuring U/SQL EJE Thin Client to Connect to the SQL Server..... | 426 |
| Configuring U/SQL EJE for MS Access, MS Query or Crystal Reports | 430 |
| Advanced Use | 432 |
| Advanced Use of U/SQL Adapters..... | 432 |
| Multi-company Support..... | 433 |
| Issues and Limits | 434 |
| Accessing the UDD System Tables..... | 436 |
| Hints and Tips | 440 |
| Troubleshooting..... | 443 |

Table Of Contents

| | |
|---|-----|
| Troubleshooting | 443 |
| Appendices | 445 |
| Appendix A - Support Information..... | 445 |
| Appendix B - Sample License Form | 446 |
| Appendix C - U/FOS data type enhancements | 447 |
| Appendix D - ACUCobol Configurable Variables..... | 448 |
| Appendix E - MFOCUS Configurable Variables..... | 449 |
| Printed Documentation..... | 451 |
| Index..... | 453 |

Introduction

Transoft U/SQL Adapters (U/SQL) provides direct Microsoft ODBC (open Database Connectivity) access to your COBOL, C-ISAM, BASIC and other non-relational data.

This enables you to use the widest possible variety of PC Windows products without changing your code or exporting your data.

- **Office Systems**

Create direct links from your data into spreadsheets or WP documents.

- **Report Writers and Business Intelligence Tools**

Use the latest GUI-based reporting and decision support products, such as Crystal Reports, Cognos Impromptu and IQ/Objects.

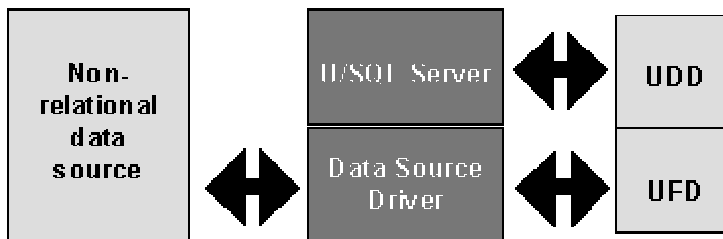
- **Windows-based Development Tools**

Modernise your applications by selectively developing and integrating modules in Microsoft Visual Basic, Delphi, PowerBuilder and so on.

The major elements of U/SQL are:

- The ODBC driver whose settings are defined by the U/SQL Administrator.
- The U/SQL Server, including optimized query planner.
- Data source drivers that access non-relational data, including specific data source drivers for ACUCOBOL Vision, Micro Focus Callable File Handler (EXTFH), C-ISAM, Business Basic ISAM, U/FOS and so forth.

Using two-level data dictionary technology, U/SQL provides a 'relational view' of your non-relational data. The Universal Data Dictionary (UDD) provides a relational description of your data and the Universal File Dictionary (UFD) describes the physical structure of the files being accessed.



The two-level data dictionary technology

Installation and Licensing

Installation and Licensing

This section describes how to install and license both **Single-tier** and **Multiple-tier** U/SQL.

- In the Single-tier model of U/SQL, both the Client and Server elements reside on the same platform, for example, a Windows 2000 based PC.
- In the Multiple-tier model of U/SQL, the U/SQL ODBC driver resides on the client platform, which is connected via a LAN to the server platform, where the U/SQL Server and Data Source Driver reside. For example, your client platform could be a Windows 95 PC and your server platform a UNIX or Windows NT Server machine.

The following sections describe:

- [Installing the Multiple-tier U/SQL server](#)
- [Installing the Single or Multiple-tier U/SQL client.](#)

Installing the Multiple-Tier U/SQL Server

Note: You must install the Multiple-tier U/SQL Server software first in order to obtain a free TCP/IP socket port number. You need to enter this when you subsequently install the U/SQL Client software.

The U/SQL Server software is available on many UNIX platforms for all data sources and on Windows NT Server and Windows 2000 Server for selected data sources including ACUCOBOL, Micro Focus COBOL and BBASIC ISAM.

This section covers:

- [Server platforms supported](#)
- [Installing the U/SQL Server Software on UNIX](#)
- [Installing the U/SQL Server Software on Windows NT.](#)

Server platforms supported

The U/SQL Server software is available on various UNIX platforms for all Data Source Drivers and on Windows NT Server 4 for selected data sources, including ACUCOBOL, BBASIC ISAM, Micro Focus COBOL.

Contact your Transoft Account Manager for a complete list of platforms and data sources supported.

Installing the UNIX U/SQL Server Software

Note:

- *If you are installing a new version of the U/SQL Server software into the same directory as a previous version, ensure that the U/SQL Server executable, **usqlsd**, is NOT running, otherwise it will not be replaced by the new version.*
- *If you are installing a release that has **U/SQL HTTP Server** (U/SQL 421 onwards) then include extra steps from the following document [Installing HTTP Server on UNIX](#)*

To install the U/SQL Server software, perform the following steps:

1. Create a base directory for the U/SQL Server software and use **tar** to copy the files from the media. For example, to install the software in the directory **/usr/usqls**, enter the following:

```
su                               : become superuser

cd /usr

mkdir usqls                       : make the usqls directory

cd usqls
```

2. Next, insert the first diskette or tape, in a suitable peripheral drive, and then issue the **tar** copy command as either:

```
tar -xv                                : if the peripheral is the default one, or
```

```
tar -xvf                                : where <device_name> is the name of the
/dev/<device_name>                       peripheral used
```

Note: If the U/SQL Server software has been supplied on more than one floppy disk, then you will be asked to insert the next disk at step 4. C., below.

3. Display or print the **README** release notice, to be found in the base directory where you installed the U/SQL Server software, for example, **/usr/usqls**. Either:

```
more README                             : to display the contents, or
```

```
lp README                               : to print the contents
```

Note: Read the Release Notice carefully before continuing with the installation as it may contain information that supersedes details given in this Help file.

4. Run the script **install.sh** in your U/SQL Server software base directory, for example, **/usr/usqls**:

```
cd /usr/usqls
./install.sh
```

This script performs the following:

- A. It checks that you are logged on as a superuser.
- B. It checks that you have TCP/IP installed and that it is enabled.
- C. If your U/SQL Server software has been supplied on more than one floppy disk, then it displays the following:

```
TRANSOFT U/SQL SERVER INSTALLATION
Extract of media not complete. Extract remaining disks.
About to read from disk device /dev/rfd0135ds18
Insert disk 2 into drive and press
  'c' to change device name
  'q' to quit
or RETURN to load disk
>c
```

Note, if you enter 'c':

```
What device do you want to extract from?
Enter a number or a device name.
  0) /dev/rfd048ds9
  1) /dev/rfd148ds9
  2) /dev/rfd0135ds18
Device? 2
```

The device names are dependent on the hardware platform. The above procedure is repeated for each additional floppy disk.

- D. It decompresses and checksums the files to ensure they are not corrupt.
- E. If the installation detects a Micro Focus environment, then the following is displayed:
The installation has detected a Micro Focus environment.

You will be informed of the Micro Focus revisions supported by the U/SQL Server executable and then offered the following option:

```
Do you wish to run a test to confirm that U/SQL Server will
operate in your environment (Y/N): Y
```

You may obtain the message:

```
The Micro Focus environment is not satisfactory for the
U/SQL Server executable included in this release. Refer to
the text file MF_LINK.TXT in the lib sub-directory for
instructions on how to re-link the U/SQL Server using your
Micro Focus environment.
```

In which case follow the instructions in the **MF_LINK.TXT** file. If the environment is satisfactory, you will obtain the message:

```
Micro Focus environment satisfactory for U/SQL Server
executable.
```

You will then be asked to select the revision of Micro Focus COBOL ISAM you are using.

```
Please select the revision of Micro Focus ISAM you are
using:
(Failure to select the correct format may result in locking
problems)
```

1. MF-ISAM Rev 3 (the Micro Focus default)
2. MF-ISAM Rev 4
3. MF-ISAM Rev 5

```
Select (1, 2 or 3):
```

Usually, you will select the Micro Focus default (1), unless you have re-linked your Micro Focus COBOL run-time to incorporate one of the other Micro Focus ISAM options.

- F. It offers a further opportunity to display or print the **README** release notice.
- G. It tests for a free TCP/IP socket port and displays the number. You must record this number as it is required when you install the U/SQL Client software.

Record the port number. If you forget this port number, you can find it in the **install.data** file in the base directory of your U/SQL Server installation, for example, **/usr/usqls**, by performing:

```
cd /usr/usqls
more install.data
```

- H. For most platforms it requests whether you wish to have the U/SQL Server started automatically at system boot time.

```
Do you wish to have the U/SQL Server start at system boot
time ? (y/n):
```

```
Answer 'y' or 'n'
```

At this point the installation of UNIX U/SQL Server software is complete. You now need to install your U/SQL Server license. Refer to the section [Installing the U/SQL Server License](#).

Installing the Windows NT U/SQL Server Software

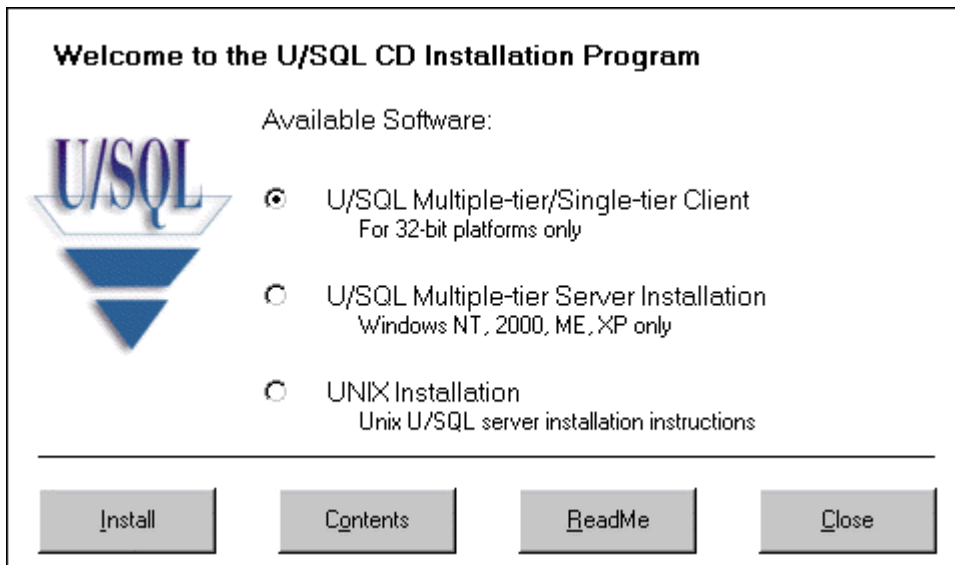
Windows NT Server

Note:

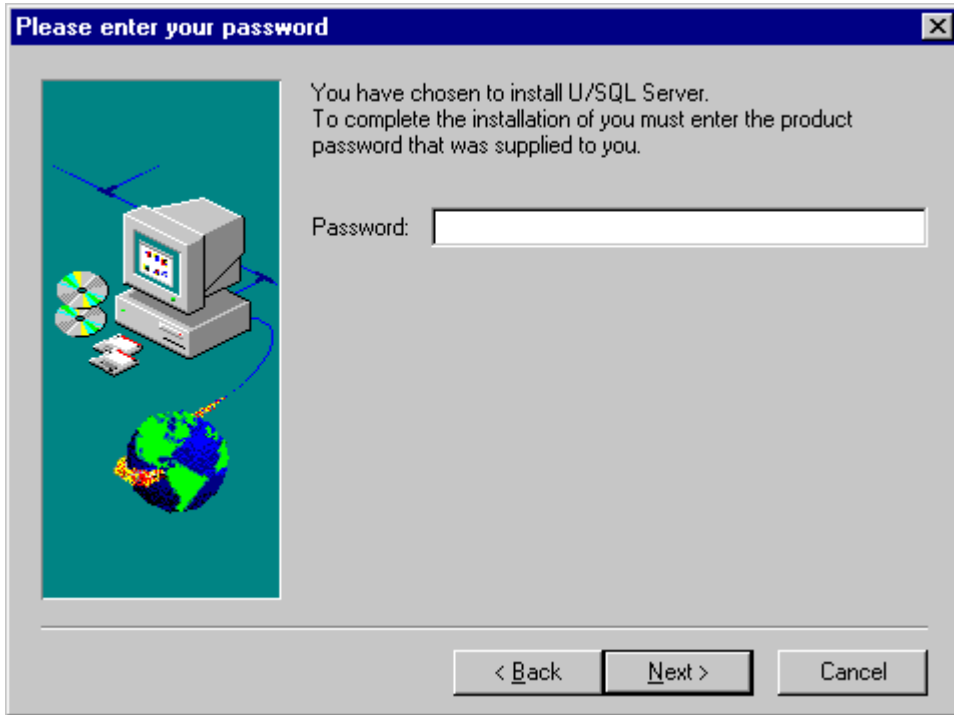
- If you are installing a release that has **U/SQL HTTP Server** (U/SQL 421 onwards) then include extra steps from the following document [Installing HTTP Server on Windows](#)

To install the U/SQL Server software on Windows NT Server 4.0 or higher, proceed as follows:

1. Ensure you are logged on with 'Administrator privileges'. Insert the U/SQL installation CD-ROM into your CD-ROM drive. Either, from Windows Explorer, run **install.exe**, or it will automatically load if 'auto insert' is set on your CD-ROM drive. The install program displays the **Welcome** dialog box:



2. Click the **ReadMe** button to view the latest Release Notice (in HTML). Click the **Contents** button for a summary of the **U/SQL CD Installation Contents** (in HTML).
3. To install U/SQL Server software, select the **U/SQL Multiple-tier Server Installation** option and either:
 - Click the **Install** button on the Welcome screen, or
 - Select the **Install** option from the **Contents** document.
4. At this point you will be requested to enter a password to confirm which U/SQL Server software you have ordered and wish to install:



The password is shown on your **Transoft Software Installation** form, supplied with the CD-ROM, along side the **U/SQL XXXXX Server** (Multi-tier) entry, for example, U/SQL ACUCOBOL Server (Multi-tier).

The password is of the form **DRVn-XXX-ABCDEFG** where **XXX** will be, for example, ACU for ACUCOBOL.

If you entered an invalid password then an error message is displayed. Click **Yes** to re-enter the password. Click **No** to abort the installation.

5. The installation tests for a free TCP/IP socket port and displays the number.

You must record this number as it is required when you install the U/SQL Client software.

The port number a server is running on can also be displayed and changed from the [U/SQL Service Manager](#) utility.

6. You will also be asked if you want the U/SQL Server to be started as an NT Service, either *automatically* each time the NT machine is started, or *manually* as required.
7. The installation of the U/SQL Server is now complete. It will have created the U/SQL Adapters program group containing:
 - U/SQL Service Manager
 - U/SQL Server Release Notes
 - unInstallShield for removing the U/SQL Server software.
8. From the Windows **Start/Programs** menu select the U/SQL program group and from the group select the **U/SQL Server Release Notes**. Ensure you read these notes carefully before continuing with the installation as they may contain information that supersedes details given in this Guide.
9. You now need to install your U/SQL Server license. Refer to the [Installing the Server License on Windows NT Server](#) section for more information.

Uninstalling U/SQL Server on Windows NT Server

Windows NT Server

To uninstall the U/SQL Server software, you must:

- Use the [U/SQL Service Manager](#) utility to select the Service to be uninstalled and click the **Uninstall Service** button. This removes Windows NT Server's own registry entries.
- Use the **U/SQL uninstallShield** to remove the U/SQL Server software.

The Next Steps

Now you can:

1. Install the U/SQL Server license as described in the [Installing the Server License on Windows NT Server](#) section.
2. Install the Multiple-tier U/SQL Client software on each user's workstation, as described in the section [Installing the Single or Multiple-tier U/SQL Client](#). Then license each client as described in the section [Installing the U/SQL Client License](#).

Installing the Single or Multiple-tier U/SQL client

System Requirements

Windows

Before you install the U/SQL Client software, check that you have the following:

- An IBM or 100% compatible PC, with 486 processor or better.
- A hard disk, with a minimum of 30 MB of disk space free, if all components are to be installed.
- A CD-ROM drive.
- The amount of PC memory required will depend on the Windows applications that are being run. Most ODBC-enabled products require substantial memory. In practice, you are likely to require a minimum of 16 MB.

For Multiple-tier versions of U/SQL:

- TCP/IP software on the PC which must be Windows Sockets based, implemented as WINSOCK-compliant. The driver is **WSOCK32.DLL** which must exist in the **\WINDOWS** directory for Windows 95 and in the **\WINNT\SYSTEM32** directory for Windows NT. Examples of compliant TCP/IPs include the TCP/IP supplied with Windows 95 and NT and the latest versions of Distinct's TCP/IP, Sun's PC-NFS and FTP's PC/TCP.

Note: *TCP/IP must be fully installed. Just copying the winsock DLL is not sufficient.*

- The PC must have a network card compatible with the TCP/IP loaded.
- A working network connection to a host with TCP/IP.
- Ensure you have installed the Multiple-tier U/SQL Server software and have noted the host name and the TCP/IP socket port number, as these are required when installing the U/SQL Client. Refer to the section [Installing the Multiple-tier U/SQL Server](#).

Contents of U/SQL Client Software

The standard shipping versions of U/SQL Client software are 32-bit based and support the following platforms:

- Microsoft Windows 95, 98 and Windows NT 4.0 or later

The U/SQL Client software consists of the following components:

- **U/SQL Administrator** - This is always installed and is used to configure the ODBC directives for each data source and the Universal Data Dictionary (UDD).
- **U/SQL Manager** - This is optionally purchased. It is only applicable to COBOL data sources and is used to maintain the UDD.
- **ODBC Driver and System files** - The appropriate ODBC Driver is determined by whether you are installing a Single-tier version of U/SQL, for example for ACUCOBOL or Micro Focus COBOL, or the Multiple-tier

version, which is the same regardless of the U/SQL Server (UNIX or Windows NT Server) and data source.

- [Win U/SQLi](#) - This Interactive U/SQL utility is useful for the development and testing of specific SQL queries and for (bulk) INSERTs, UPDATEs or DELETEs to your data. In addition, it provide a facility for exporting and importing UDDs. This facility is used to upgrade UDDs and can be used to correct errors that may have been encountered in a UDD.
- [License Tool](#) - The License Tool is used to install your U/SQL Client license.
- [TestNet](#) - This utility is used to troubleshoot your ODBC Winsock and network compliance. It is most helpful for troubleshooting Multiple-tier installations.
- [Books Demonstration](#) - This is a sample Visual Basic application.
- [System & File Information utility](#) - This utility provides information on the System, ODBC files and Data Source Drivers installed.
- **Release Notice** - This can be viewed and printed. Ensure you read the Release Notice carefully as it may contain information not contained in the Online Help file.

The U/SQL Client software gives you the option to be selective in the components that are installed. These are grouped by **Compact**, **Typical** or **Custom**, as follows:

| | |
|----------------|--|
| Compact | Installs the U/SQL Administrator, License Tool, Online Help file, Release Notice, ODBC Driver and System files. |
| Typical | Installs all of Compact plus the Win U/SQLi utility and the Books Demonstration application. Typical will be the usual installation selection, unless the U/SQL Manager is to be installed (when Custom is selected). |
| Custom | Installs as default the same contents as Typical. Extra options are the U/SQL Manager and its Creating a COBOL Dictionary manual, if purchased for COBOL data sources only, and TestNet. It is also possible to optionally select most of the components individually. |

Note: *The U/SQL Manager is only available as a 32-bit product and you must select it from the options in the **Custom** installation.*

To install the software refer to the section [Installing the Client Software](#).

Installing the Client Software

To avoid possible U/SQL Client installation problems:

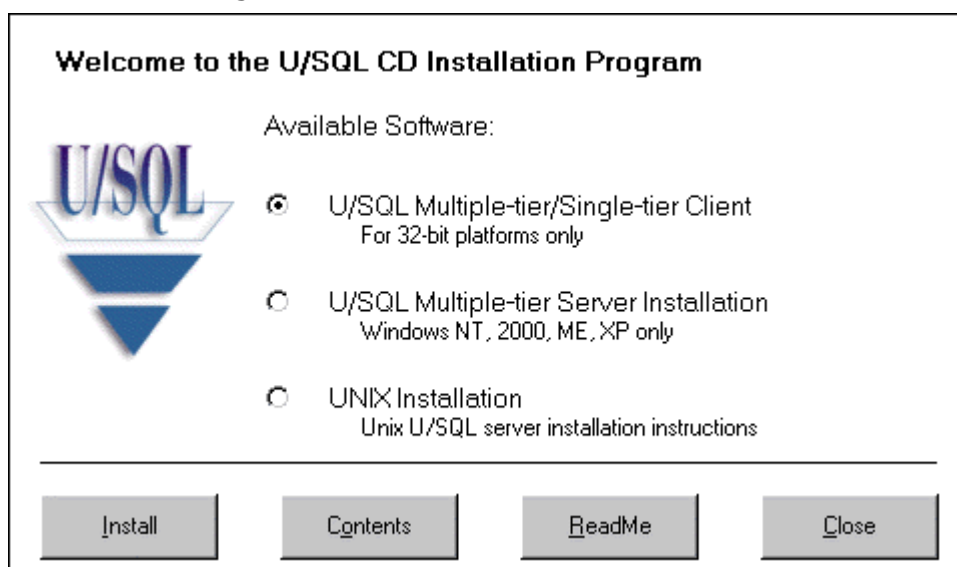
- Re-start Windows before installing to ensure no tasks are memory resident that could affect installation, this should include shutting down even the Microsoft Office Toolbar.
- When installing on Windows NT ensure you are logged on with 'Administrator privileges'.

If you have problems after installation, such as Win U/SQLi will not connect to a data source, then reboot your PC and re-install the U/SQL Client software. First de-install the original version as described in the section [De-installing the U/SQL Client Software](#).

Windows

To install the U/SQL Client software proceed as follows:

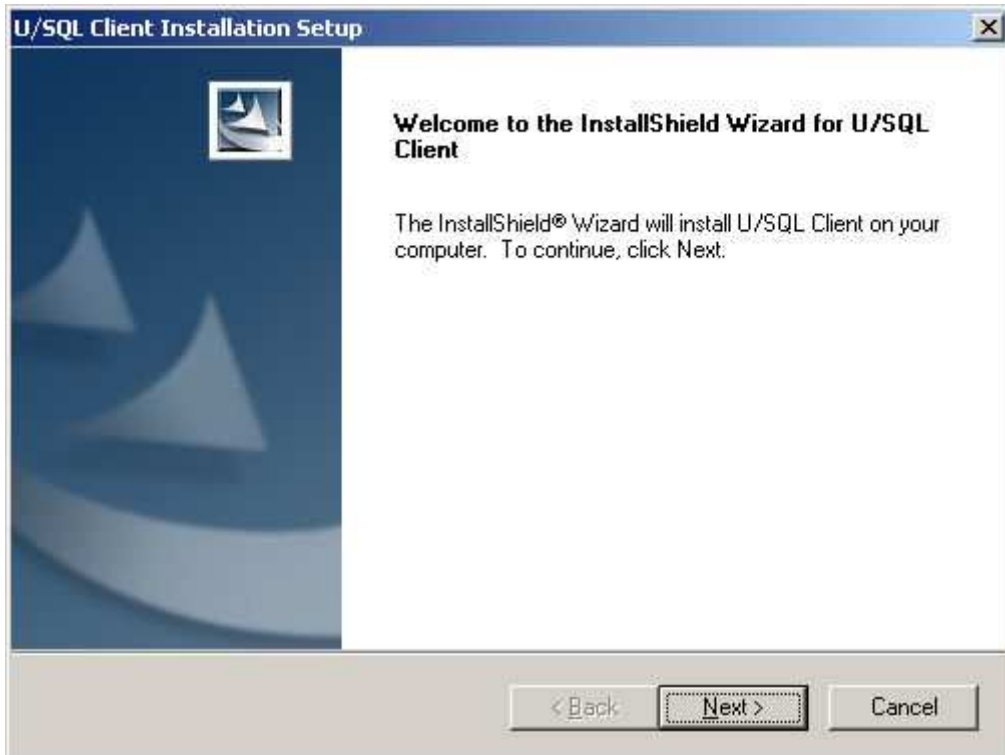
1. Insert the U/SQL installation CD-ROM in your CD-ROM drive. Using Windows Explorer, run **install.exe**, or it will automatically load if 'auto insert' is set on your CD-ROM drive. The install program displays the **Welcome** dialog box:



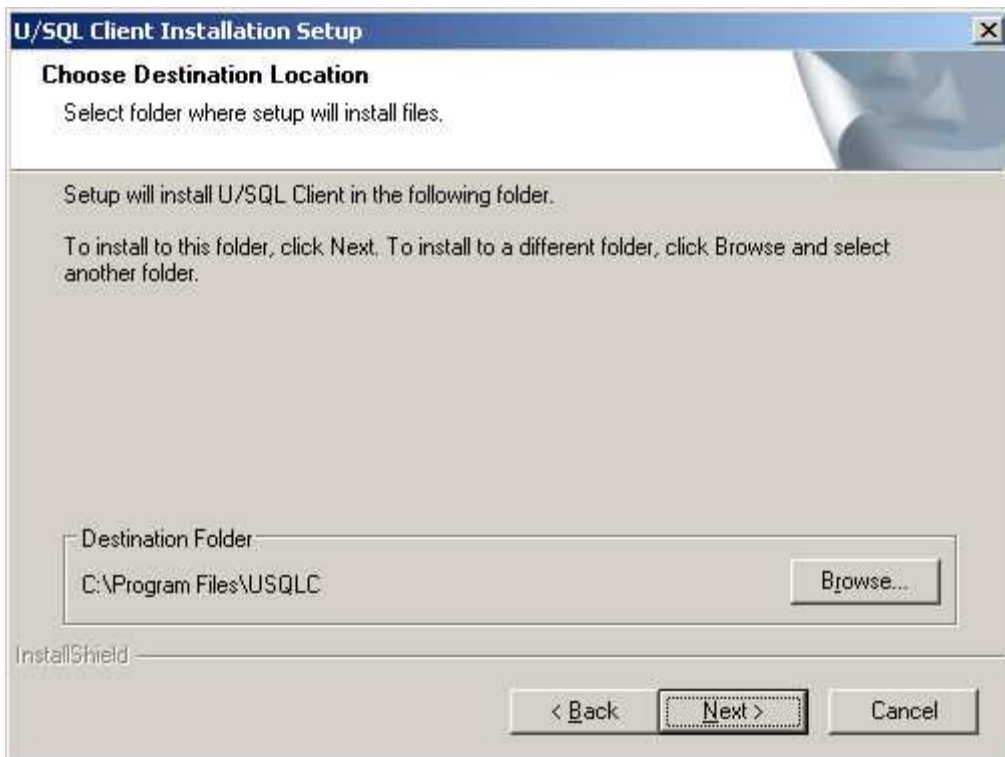
2. Click the **ReadMe** button to view the latest Release Notice (in HTML). Click the **Contents** button for a summary of the **U/SQL Client CD Installation** contents (in HTML).

In addition to installing the U/SQL Client software, you can also install **Microsoft ODBC 3.0 Driver Manager components**. These are required for some of the latest ODBC-enabled products.

3. Select the **U/SQL Multiple-tier/Single-tier Client** software option and click the **Install** button.
4. The installation prepares the InstallShield Wizard that will guide you through the installation process. The **Welcome** dialog box is displayed:

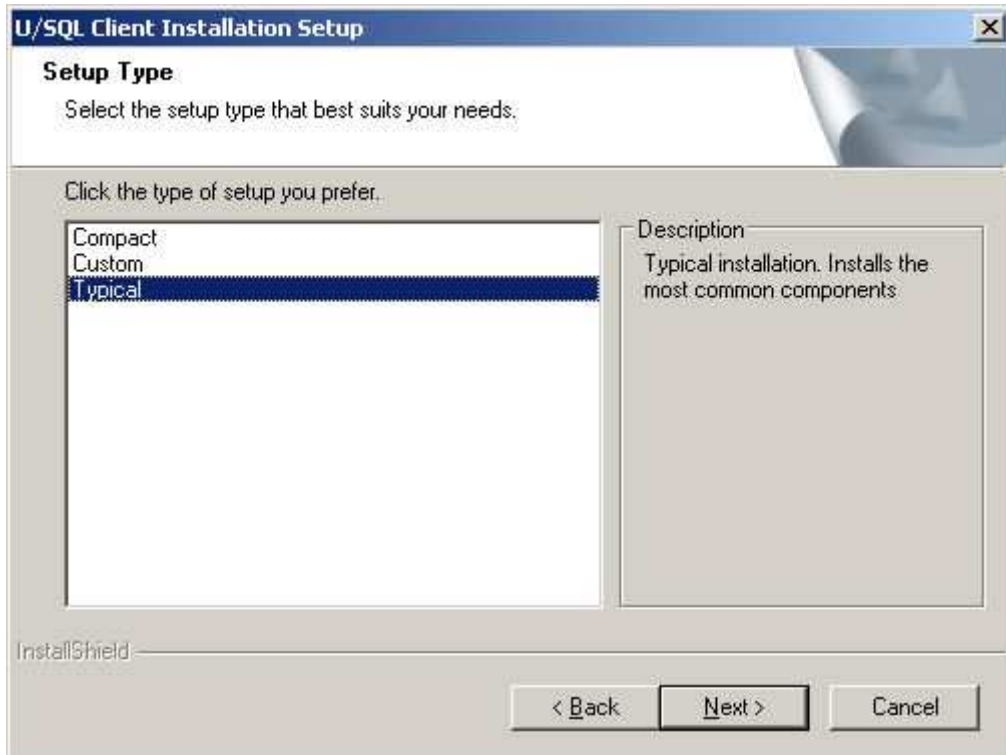


5. Click **Next**. The **Choose Destination Location** dialog box is displayed:



Decide, using the browser where you want to install the U/SQL Client. The default location is **C:\Program Files\USQLC**. Click **Next**.

6. The **Setup Type** dialog box is displayed:



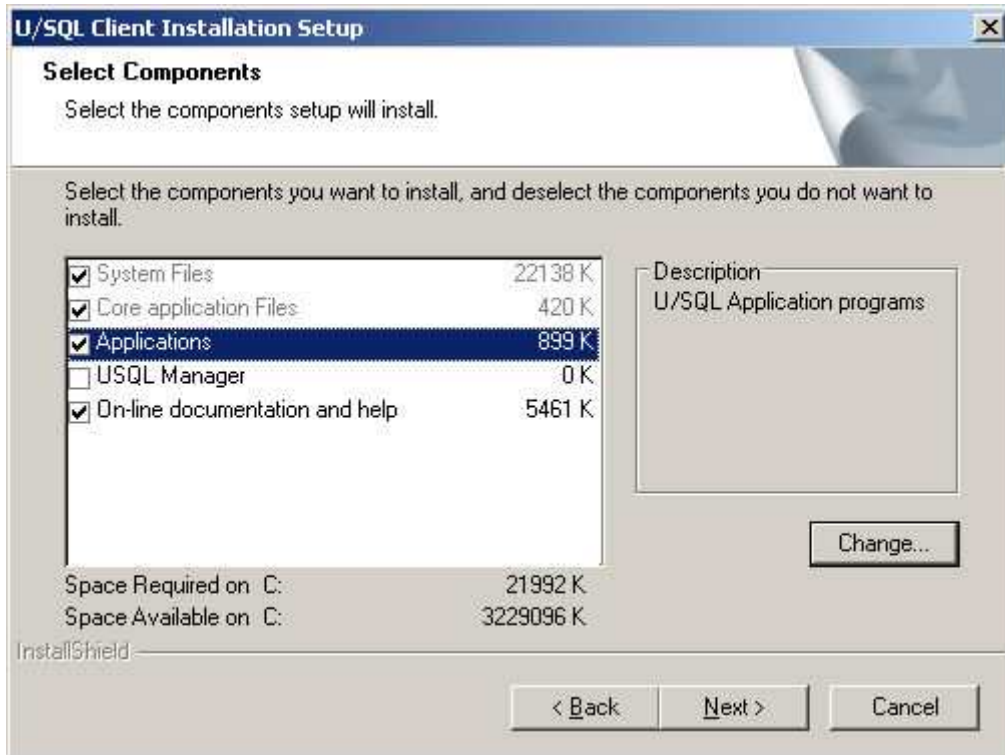
Note: *If you have optionally purchased the **U/SQL Manager**, for COBOL data sources, you must select the Custom install.*

Choose one of the following options (the default is Typical):

Compact - Installs the U/SQL Administrator, License Tool, manuals, Release Notice, ODBC Driver and System files.

Typical - Installs all of Compact plus the Win U/SQLi utility and the Books Demonstration application.

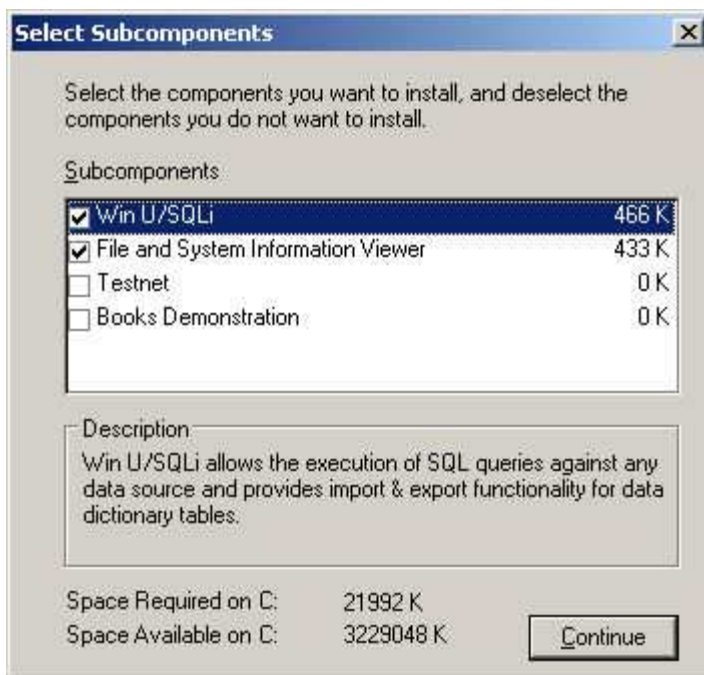
Custom - Installs as default the same contents as Typical. Extra options are the U/SQL Manager and its Creating a COBOL Dictionary manual, if purchased for COBOL data sources only, and TestNet. It is also possible to optionally select most of the components individually.



Custom is also be used to install any component that was not installed in the original U/SQL Client software installation. The installation process is repeated but, for example as you have been having network problems, you only select to install TestNet.

If you wish to have a Dual-tier installation, that is have both Single and Multiple-tier ODBC Drivers installed so you can access both local PC based data and remote server based data, then you would use the Custom option to select only ODBC Drivers to install the second ODBC Driver.

Click the **Change** button to allow the selection of **Sub-components**. The **Select Sub-components** dialog box is displayed:



You can decide that you do not require, for example, Win U/SQLi to be installed.

7. At this point you will be requested to enter a password to confirm the ODBC Driver you have ordered and wish to install:

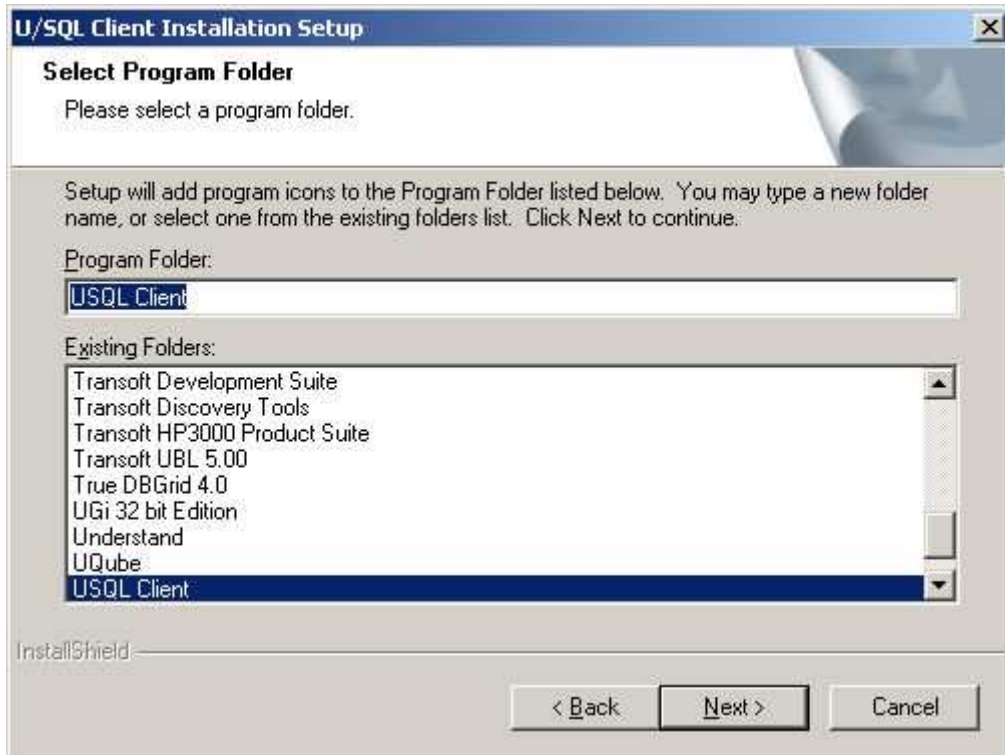


The password is shown on your Transoft Software Installation form, supplied with the CD-ROM, along side the U/SQL ODBC Driver entry, for example, U/SQL ODBC Driver (32-bit MT).

The password is of the form **DRVn-XXX-ABCDEFGH** where **XXX** will be, for example, ACU for Single-tier ACUCOBOL and MF for Micro Focus COBOL; MT for Multiple-tier. Other Single-tier versions are also available.

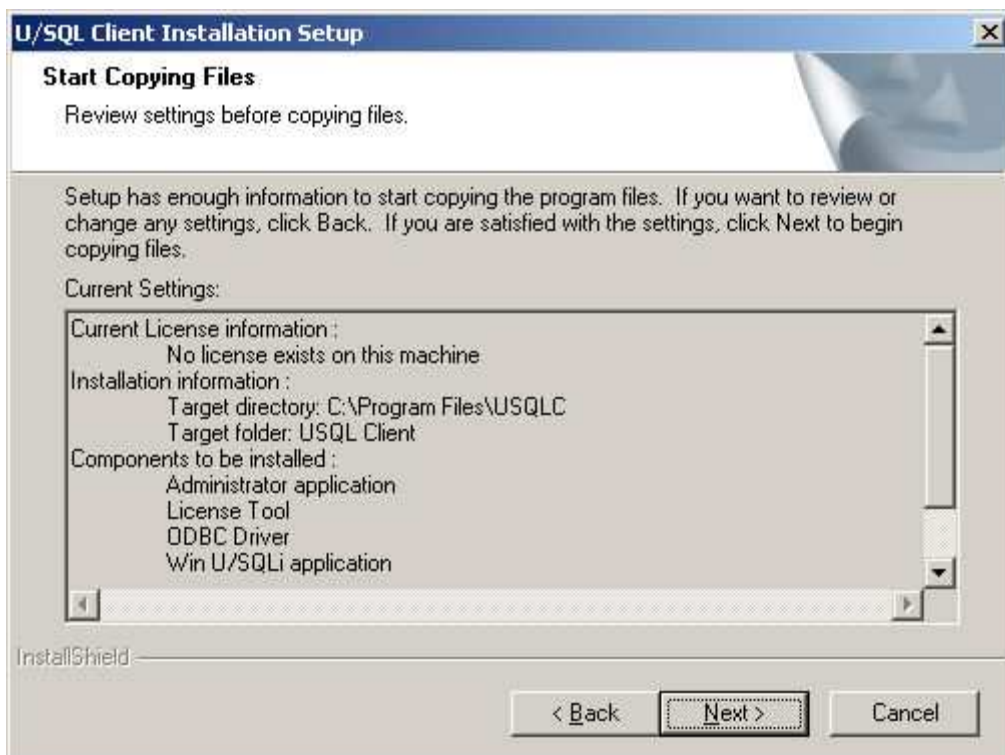
If you enter an invalid password an error message is displayed. Click **Yes** to re-enter the password. If you click **No**, then the ODBC driver will fail to be installed and it is recommended you abort the installation.

8. Next select your Program Folder, the default is **USQL Client**:



Click **Next**.

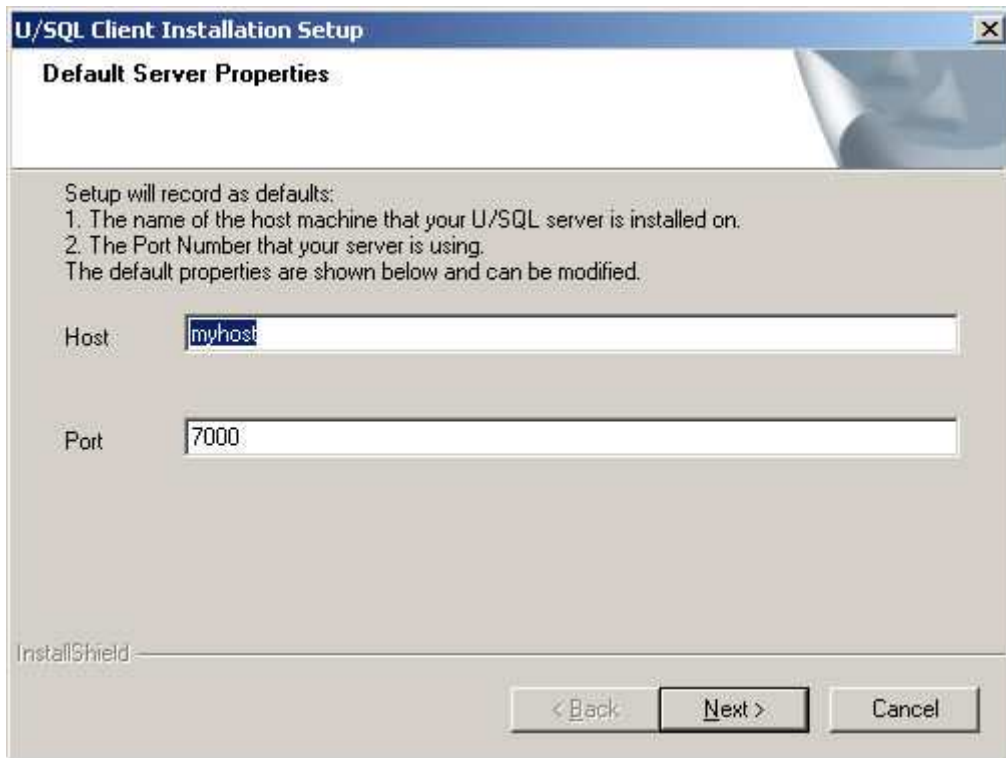
- At this point the installation displays what it intends to install from the selection you have made:



If you want to make any changes before proceeding with the installation click the **Back** button. Otherwise, click **Next** to start installing the software.

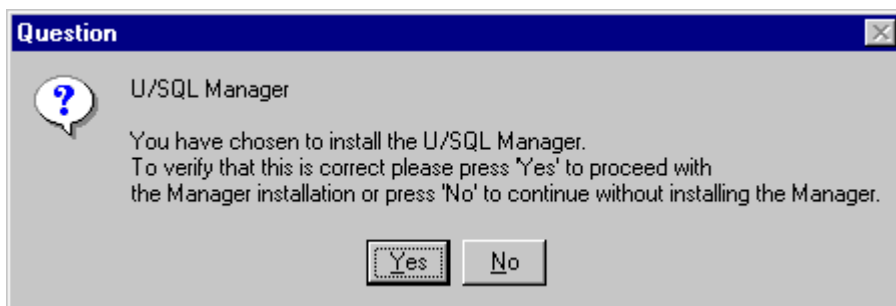
- Once the U/SQL Client software has finished installing, then if you are installing the Multiple-tier ODBC Driver, you will be requested to enter the

Host name and **Port Number** of the UNIX or Windows NT Server on which you have installed the U/SQL Server software:



Click **Next**.

11. After the installation of the ODBC Driver and if you have selected to install, in step 6, the **U/SQL Manager** for COBOL data sources, then you will be presented with the following message:



Click **Yes** to continue with the installation. If for any reason you do not want to install the U/SQL Manager, then click **No**.

12. Unless you are upgrading the U/SQL Client software and therefore already have a license installed, you will be presented with the **License Tool** and **License Installation Instructions**. You can install the U/SQL Client license as part of this software installation process, or any time later, as described in the section [Installing the U/SQL Client License](#).
13. You will be presented with the "USQL Setup complete" screen. Click "Finish" to exist the installation.

De-installing the U/SQL Client Software

Windows 95 & NT

The U/SQL Client software may be de-installed as follows:

1. Remove the U/SQL Client ODBC driver using the [U/SQL Administrator](#). Select the **Drivers** dialog tab, highlight the Transoft ODBC Driver to be removed and click **Remove**.
2. Remove the U/SQL Client application software using the Windows **Control Panel**. Select **Add/Remove Programs**, select the **Install/Uninstall** tab, and then the **U/SQL Client Rev x.xx** entry. Follow the on-screen instructions.

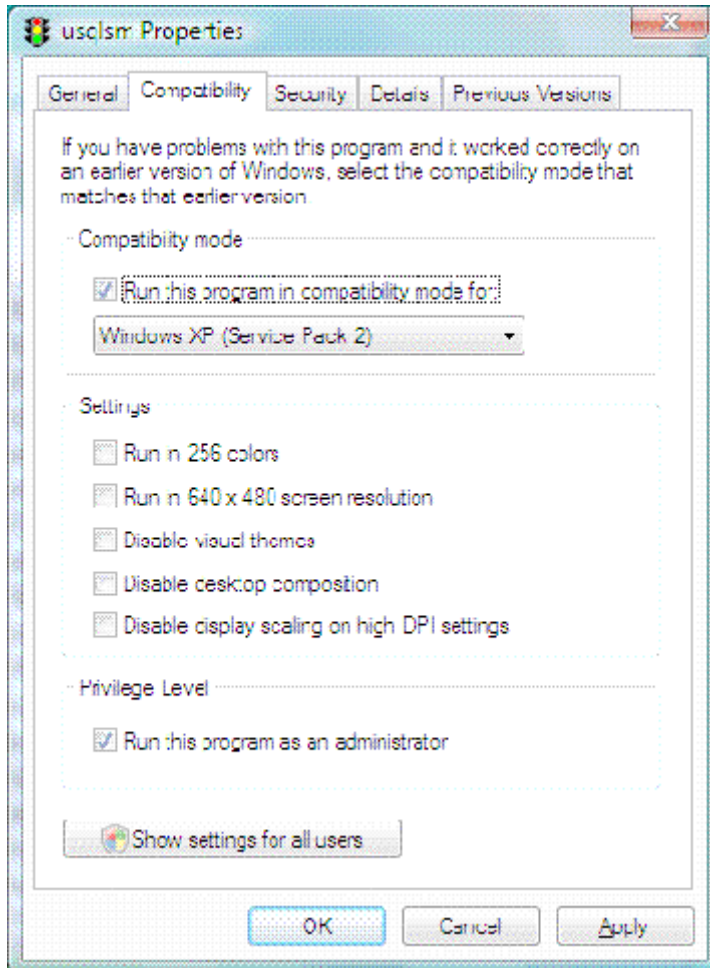
The Next Step

In order to install the U/SQL Client license refer to the section [Installing the U/SQL Client License](#).

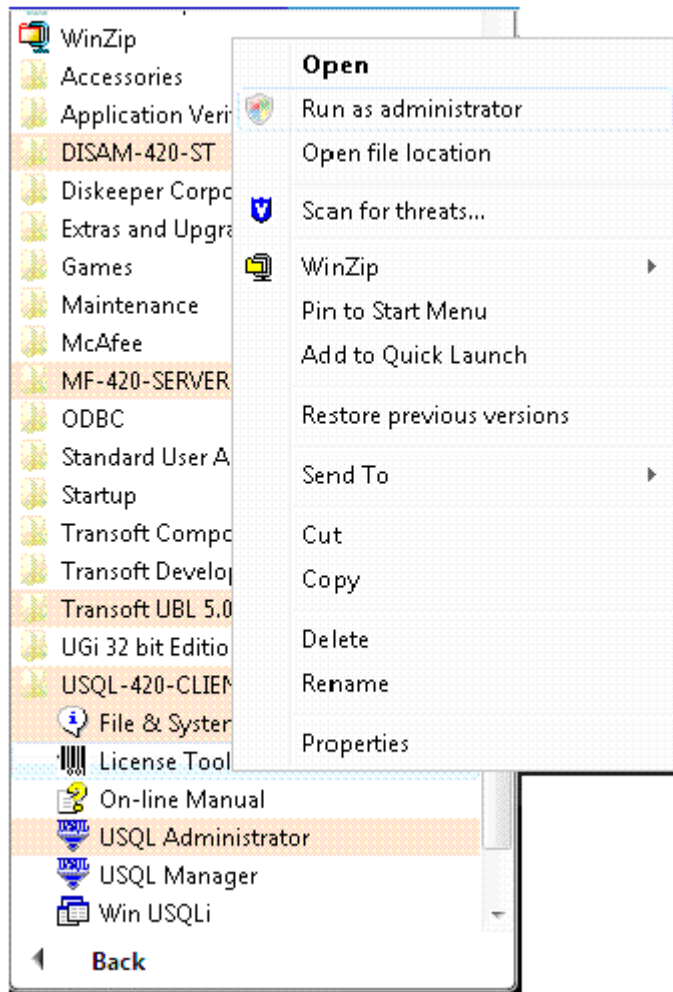
Installing and Licensing Vista Business edition

Additional installation instructions for installing the U/SQL Server on Vista Business:

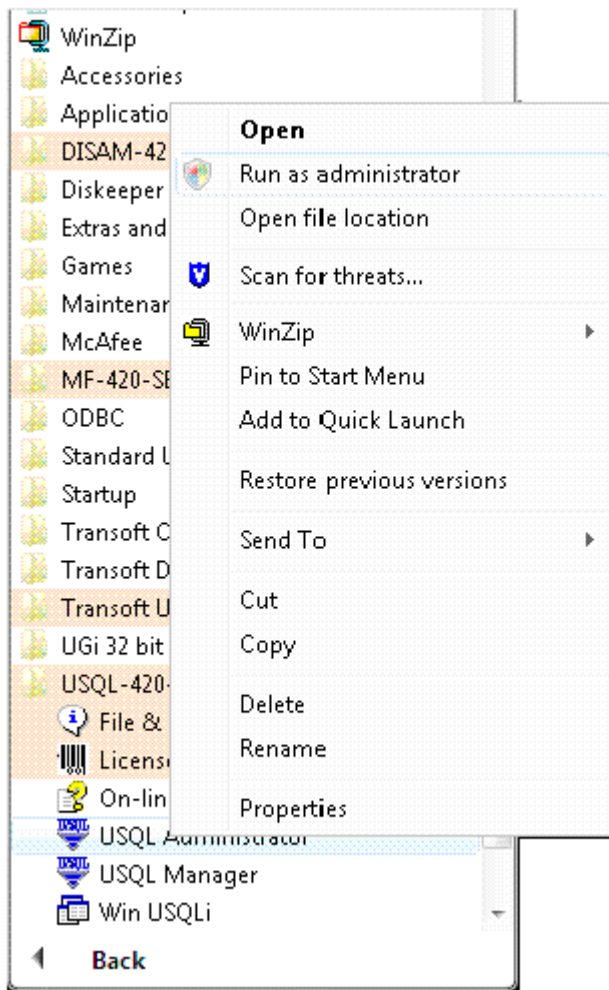
- After installing the U/SQL server, locate the service manager executable (usqlsm.exe), right click on the usqlsm.exe, select properties->compatibility and check the "Run this program as administrator" option in the 'Privileges' section.



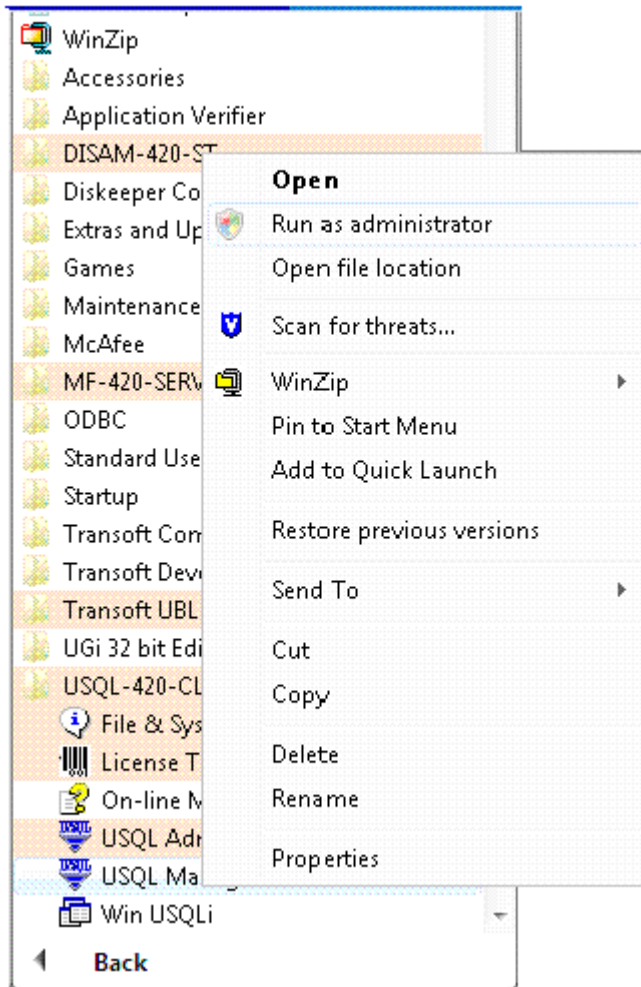
- When licensing (i.e. initial install or subsequent upgrade) the U/SQL Client, you must run license tool as Administrator.



- When adding, configuring or deleting a System DSN, you must run the U/SQL Administrator as 'Administrator' i.e. right click on the target > select 'Run As Administrator'.



- When adding, configuring or deleting a User DSN, you can run the U/SQL Administrator as an ordinary user.
- When using the U/SQL Manager you must run it as 'Administrator' i.e. right click on the target > select 'Run As Administrator'.



- Please also note that Win USQLi can be run as an ordinary user.

Installing HTTP Server on Windows

U/SQL Client Browser on Windows

Introduction

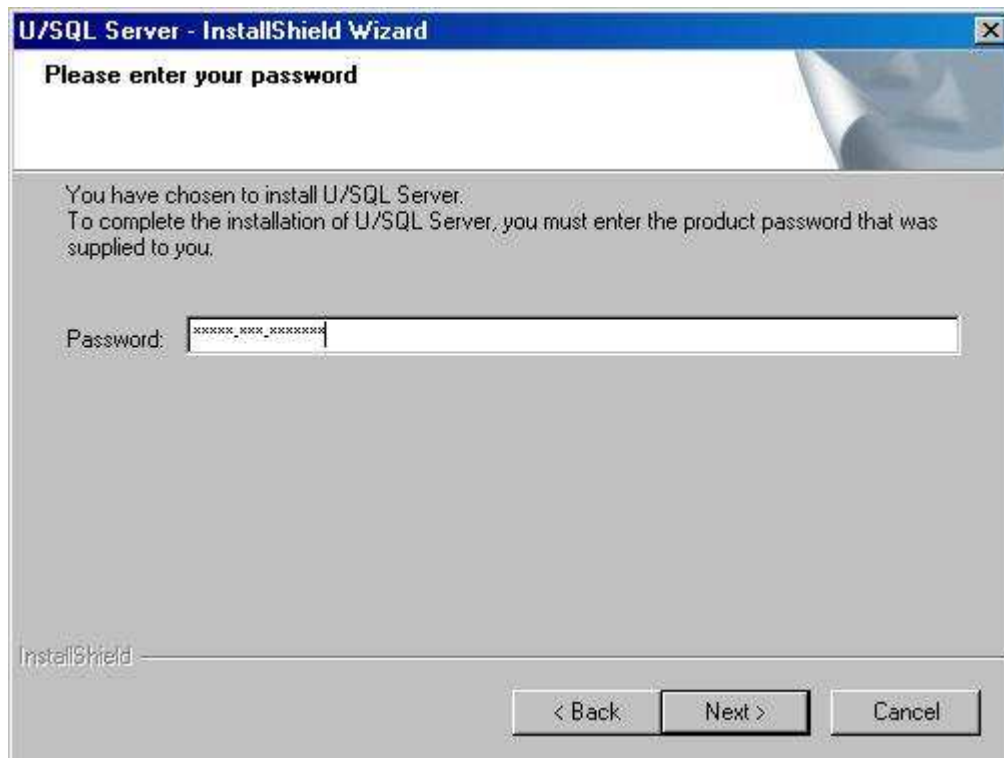
Transoft's U/SQL HTTP Server, introduced in engine revision 4.20.0100, extends the query facilities available in U/SQL to allow remote HTTP access to your U/SQL Server instance.

Accessed via your Internet browser, the HTTP Server is a quick and easy way to interrogate your UDD and data.

Installation

The HTTP Server service is an optional element of the Windows U/SQL Server installation.

After entering the product password



The installation script will generate the following prompt:



The U/SQL and HTTP services are installed as separate entities; after installation, if you look in your list of services you should see two distinct service entries:



Configuration

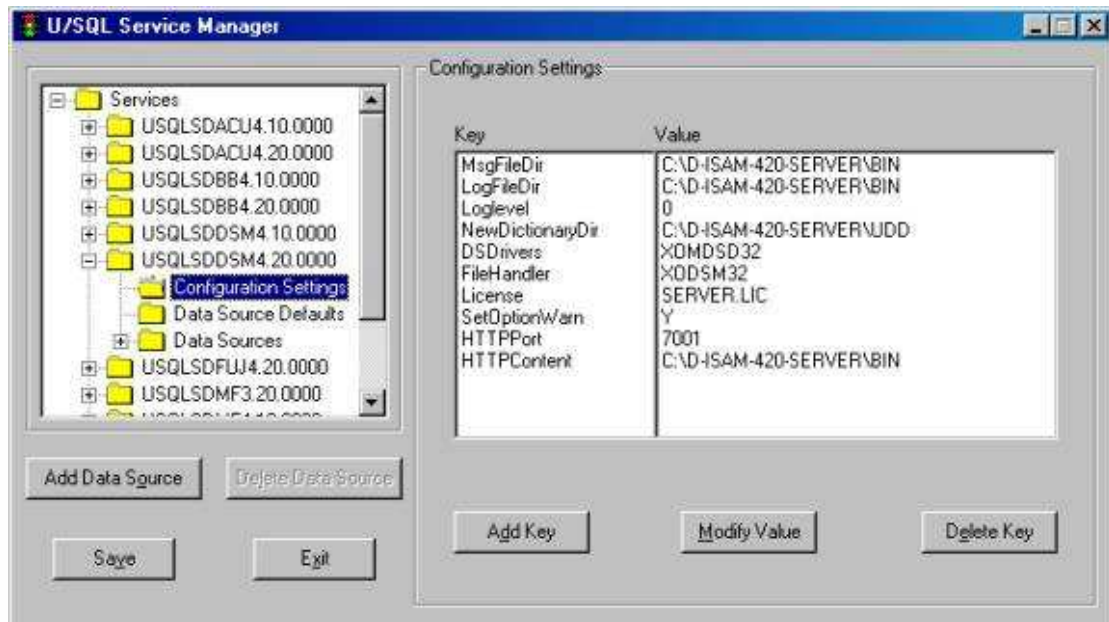
The HTTP Server relies on two **[Configuration Settings]** directives in order to determine the location of the HTTP content files and the service port. Added and set manually, the directives have the following values and format:

HTTPPort=<port number>

and

HTTPContent=<full path to where the home.xml and usql.xsl files reside>

These directives are set in the U/SQL Service Manager



These two directives must be set prior to starting the HTTP service.

Note: please ensure the U/SQL Server install is run-as Administrator as the home.xml and usql.xsl files are copied to the U/SQL bin directory as part of the installation of the HTTP Server.

Getting started with the HTTP Server

The HTTP Server is an optional start up element; when starting the U/SQL Server service you will be presented with the following:



If you decline this option, the U/SQL HTTP Server service will not start.

You can stop the HTTP Server either via the U/SQL Service Manager (stopping the U/SQL Server service will stop the HTTP Server) or **Control Panel > Services** (this gives the option of a selective shutdown of the running U/SQL services)

Accessing your data via the HTTP Server

The HTTP server is accessed via your Internet browser. To access the service you must use either of the following URL formats

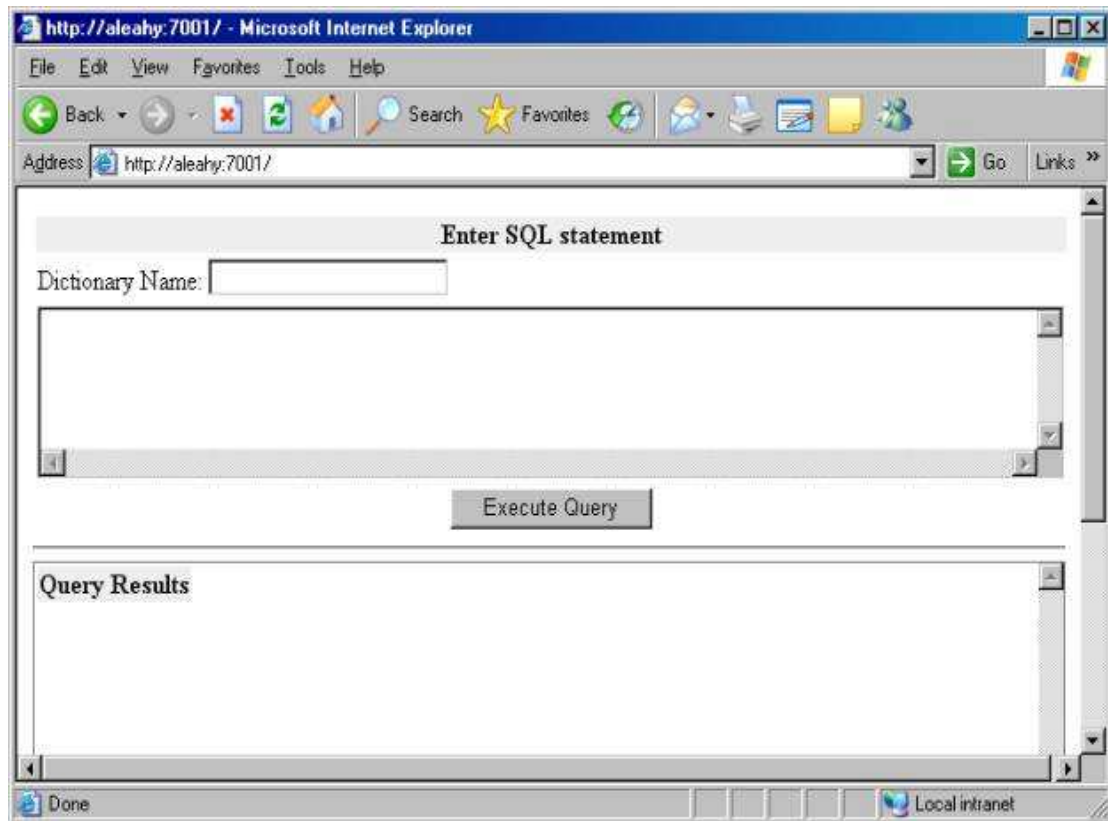
<http://machine:port>

or

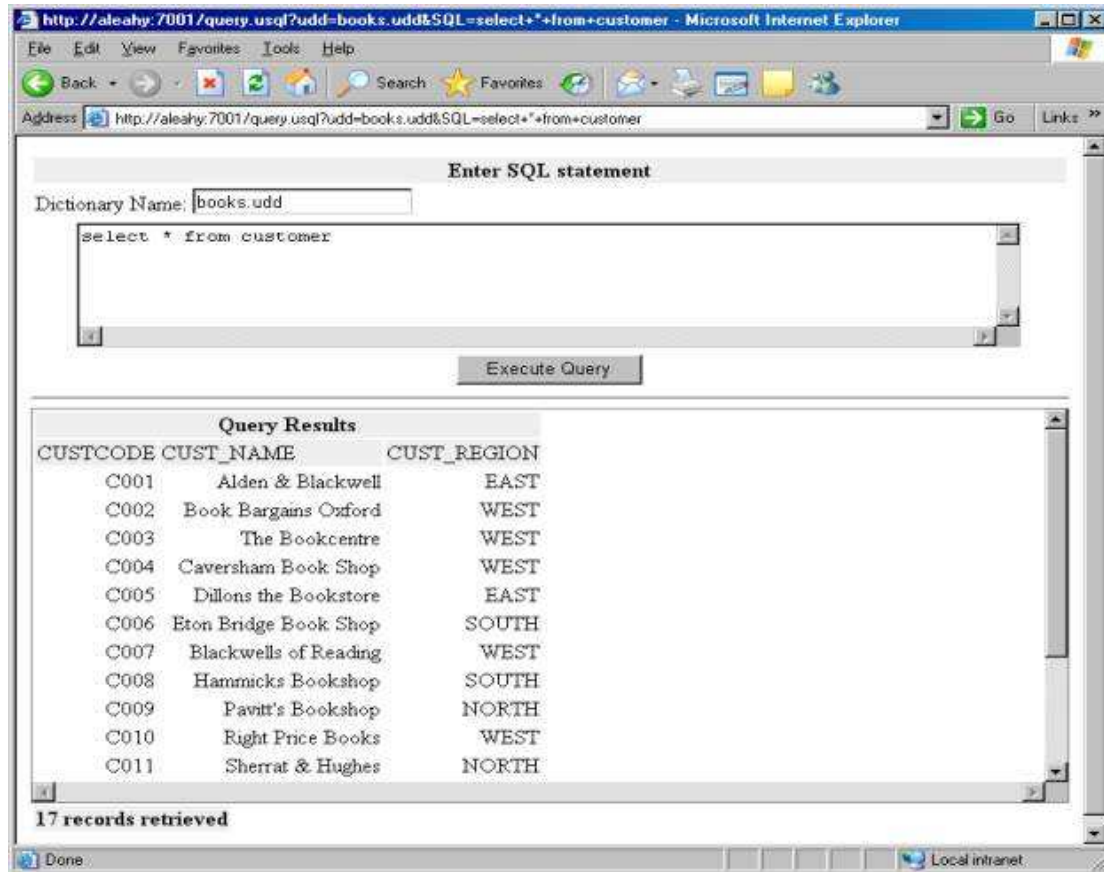
<http://machine:port/home>

No other path is valid.

When you access the service you are presented with the following display:



After entering the relevant UDD details, you can run 'typical' ANSI SQL statements e.g. SELECT, UPDATE and DELETE statements.



Diagnostic logging

In terms of activity logging, HTTP server log messages are logged into usqlcs_http.log (located in the U/SQL bin directory).

A successful start-up and query return should produce something similar to the following:

```
DBG: 26/Mar/07 12:24:56 SYSTEM 3640 httpserver.c 135:
--- HTTP server[1.00.0000] started on port 7001.
DBG: 26/Mar/07 12:24:59 SYSTEM 3796 httpserver.c 149:
--- New HTTP client 232 .
DBG: 26/Mar/07 12:24:59 SYSTEM 3796 httpserver.c 239:
--- HTTP server responded 200.
DBG: 26/Mar/07 12:24:59 SYSTEM 3796 httpserver.c 165:
--- HTTP client 232 gone.
```

HTTP client messages are logged into usqlcs.log.

Limitations

The U/SQL HTTP Server query interface has a number of limitations, namely:

- You cannot create a new UDD.
- You cannot set Multi-tier security or 'Grant & Revoke' security through the web interface. You can, however, log-in to existing datasources that have been configured to use these security options.
- Only one query can be run at one time.

Installing HTTP Server on UNIX

U/SQL HTTP Client Browser on UNIX

Introduction

Transoft's U/SQL HTTP Server, introduced in engine revision 4.20.0100, extends the query facilities available in U/SQL to allow remote HTTP access to your U/SQL Server instance.

Accessed via your Internet browser, the HTTP Server is a quick and easy way to interrogate your UDD and data.

Installation

The HTTP Server executable (usqlhttp) is installed, along with associated files, by default during the U/SQL Server installation.

To confirm installation, check in your U/SQL bin directory for the following files:

usql.xsl

home.xml

usqlhttp

The U/SQL and HTTP executables are installed as distinct entities; use of the HTTP Server is optional.

Configuration

The HTTP Server relies on two **[Configuration Settings]** directives in order to determine the location of the HTTP content files and the HTTP Server port. Added and set manually, the directives have the following values and format:

HTTPPort=<port number>

and

HTTPContent=<full path to where the home.xml and usql.xsl files reside>

These directives are set in the usqlsd.ini situated in the U/SQL bin directory:

[Configuration Settings]

DefaultServer=sgsun3

DefaultPort=7001

HTTPPort=7002

HTTPContent=/home/proj/support/aleahy/httpserver/bin

These two directives must be set prior to starting the U/SQL Server.

Getting started with the HTTP Server

The HTTP Server is an optional start up element; when starting the U/SQL Server you will be presented with the following:

***The U/SQL Server has started on port '7001'.
Start HTTPServer on the default port number 7002 (y/n):***

Once started, the HTTP Server will remain in a 'listening' state:

***The U/SQL Server has started on port '7001'.
Start HTTPServer on the default port number 7002 (y/n): y
listening on port 7002***

You can decline the HTTP start-up option by simply following the prompts:

***Start HTTPServer on the default port number 7002 (y/n): n
Start HTTPServer on the next available port number 7000 (y/n): n
Enter the port number on which you would like to start the HTTPServer
(or type 'x' to EXIT) : x
\$***

If you want to stop the HTTP Server you will need to shutdown the U/SQL Server via the stop_serv.sh.

If you previously declined the HTTP Server start-up option, and wish to start it, you will need to shutdown the U/SQL Server and run the start_serv.sh again.

Accessing your data via the HTTP Server

The HTTP server is accessed via your Internet browser. To access the service you must use either of the following URL formats

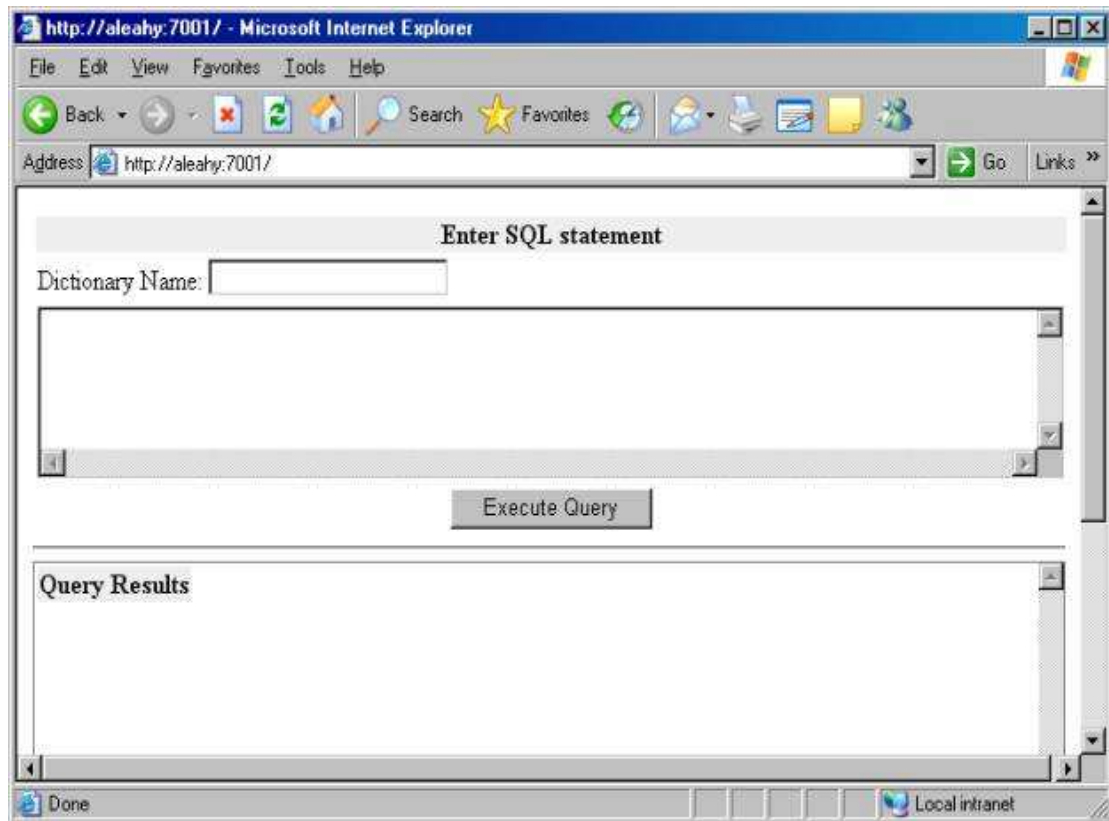
<http://machine:port>

or

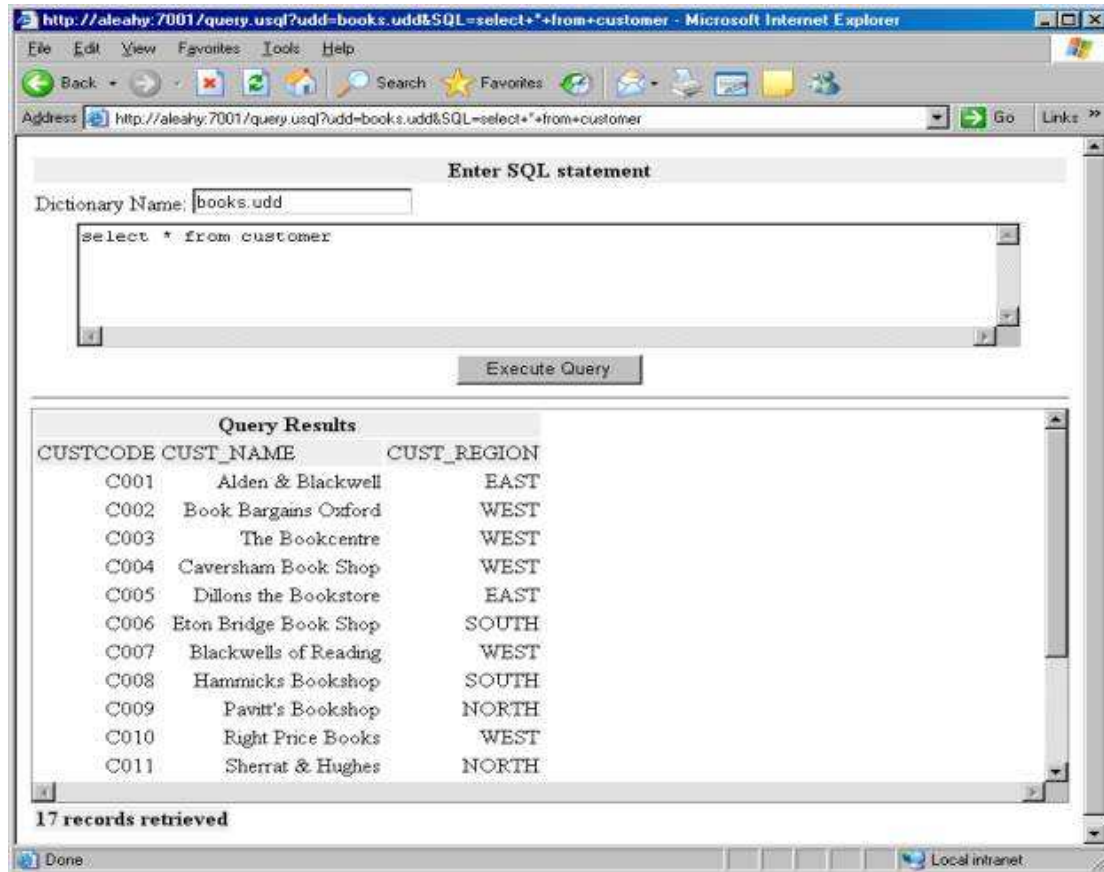
<http://machine:port/home>

No other path is valid.

When you access the service you are presented with the following display:



After entering the relevant UDD details, you can run 'typical' ANSI SQL statements e.g. SELECT, UPDATE and DELETE statements.



Diagnostic logging

In terms of activity logging, HTTP server log messages are logged in the standard usqlcs.log (located in the U/SQL bin directory).

A successful start-up should, at log level 4, produce something similar to the following:

```
CON: 27/Mar/07 12:05:06 aleahy 23156 unixserv.c 342:
--- usqlsd started on port 7005.
DBG: 27/Mar/07 12:05:10 aleahy 23167 httpserver.c 135:
--- HTTP server[1.00.0000] started on port 7002.
DBG: 27/Mar/07 12:05:24 aleahy 23167 httpserver.c 149:
--- New HTTP client 5.
```

Limitations

The U/SQL HTTP Server query interface has a number of limitations, namely:

- You cannot create a new UDD.
- You cannot set Multi-tier security or 'Grant & Revoke' security through the web interface. You can, however, login to existing data sources that have been configured to use these security options.

- Only one query can be run at one time.

Licensing

Licensing

This section describes how to install the U/SQL Client license and the U/SQL Server license. In addition, it describes the implications of using U/SQL with the installed licenses and the possible error conditions.

U/SQL Revision 3, or above, requires a license on each user's PC for both Single and Multiple-tier, and for Multiple-tier a further license must be installed on the server. The U/SQL Server license controls the number of concurrent users connected. The licenses consist of License Codes supplied on a License Form by your supplier (see sample in [Appendix B](#)).

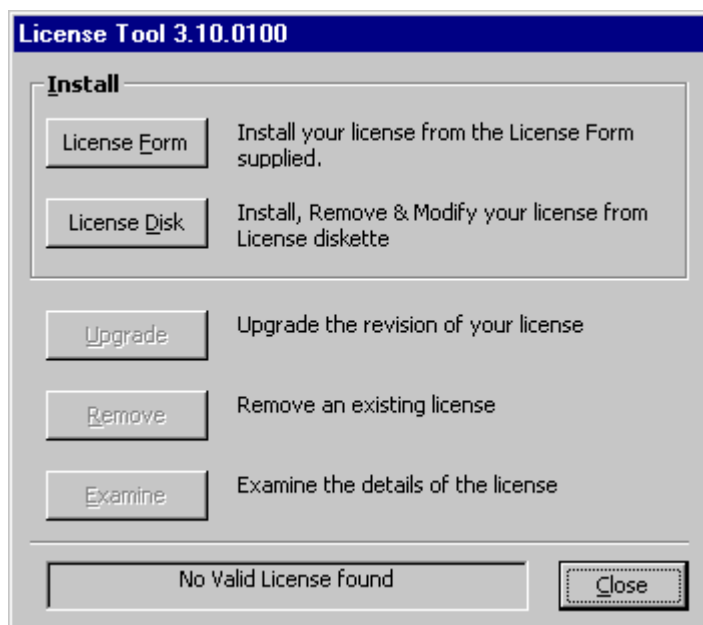
For evaluations and emergencies, for example, where the License Form containing your License Codes is mislaid following a disk crash, you can create Temporary Unlicensed Copies for 30 days. This applies to both the U/SQL Clients and the U/SQL Server.

This section covers the following topics:

- [Installing the U/SQL Client License](#)
- [Installing the U/SQL Server License](#)
- [Starting U/SQL with Licensing](#).

Installing the U/SQL Client License

If you are installing the U/SQL Client software for the first time, then the **License Tool** is displayed:



It is not essential to install your license during the U/SQL Client software installation. It can be undertaken at any time using the **License Tool** found in the U/SQL Client program group. (The License instructions are only displayed during software installation).

Also the U/SQL Client software may be used as a Temporary Unlicensed Copy for evaluation purposes for a 30 day period, without installing any license. See the section [Creating a Temporary Unlicensed Copy](#).

To license the U/SQL Client you enter an unique License Code on each user's PC from a supplied **License Form** (see sample in [Appendix B](#)). This step is described in detail in the next section.

Entering a License Code

You will have been provided with a **License Form** containing a set of unique **Client License Codes** for the number of U/SQL Clients (that include the ODBC Driver and the U/SQL Administrator) purchased. Refer to the Sample License Form in [Appendix B](#). For example, if you have ordered a four user license, there will be four unique **U/SQL Client License Codes** on the **License Form**.

One of the unique Client License Codes is installed on each user's PC.

From the **License Tool**, click the **License Form** button. The **Install License** dialog box is displayed:

Enter **Your Name** and **Your Organization**, which must be the same Organization that is shown on the License Form, followed by one of the unique License Codes which determines how the U/SQL Client software will operate.

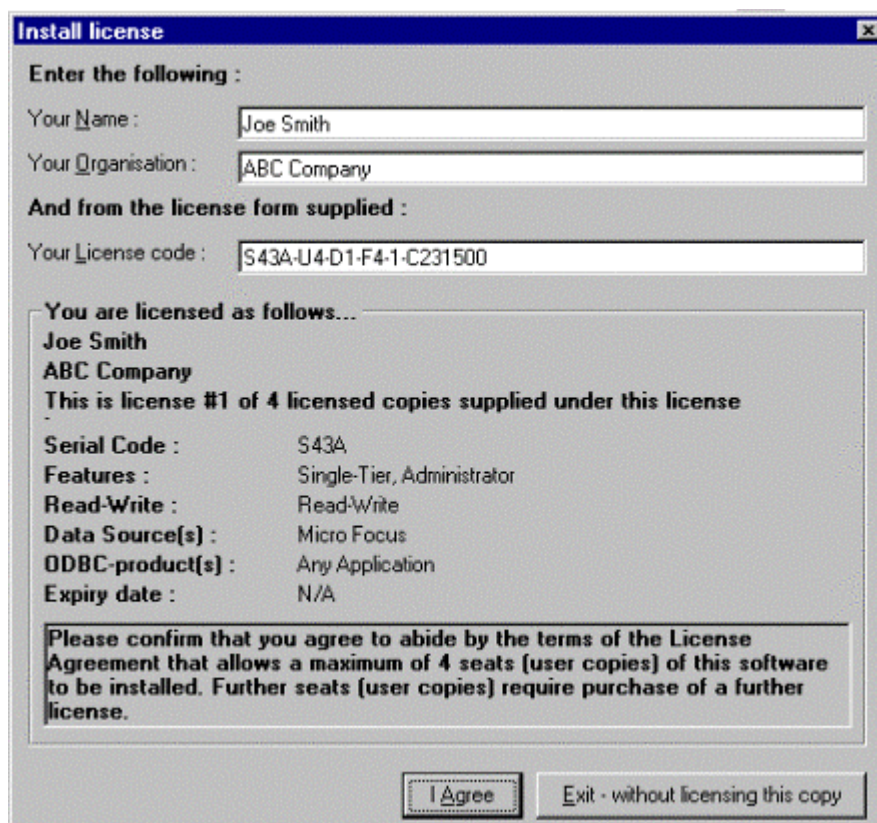
Note: *It is a good idea to register the name of the user of each License Code on the License Form for future reference. This is in case the License Code needs to be re-installed if the disk 'crashes' and to ensure that each code is used only once.*

A sample License Code is **S43A-U4-D1-F4-1-C231500**, where:

- S43A** The unique serial code for the license which MUST match the serial code on the Multiple-tier U/SQL Server License Code.
- U4** The total number of U/SQL Clients copies to be installed, in this case 4.
- D1** The Data Source Driver code, for example, Micro Focus COBOL.
- F4** The features code, determining Single or Multiple-tier, read-only or read write, U/SQL Administrator or Manager allowed, and so on.
- 1** The copy number of this license; in this case 1 of 4 (the maximum copies to be installed).
- C231500** The checksum that check all the parts of the License Code and that the correct Organization name is entered.

Note: *The checksum can be an alphanumeric code; CA2X16GDY.*

After entering **Your Name**, **Your Organization** and the **License Code** click **Install**. If an incorrect Organization / License Code combination is entered an error message is displayed. You can correct the entries or click **Exit**. If a valid combination is entered, then the following dialog box showing the details of the installed license is displayed:



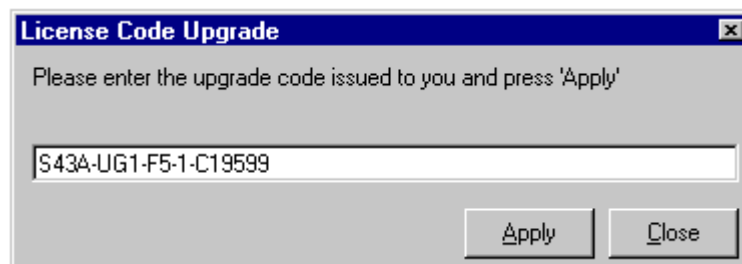
Read the terms of the License Agreement and if you agree to them click the **I Agree** button. Finally click the **Close** button to complete the licensing process.

Upgrading a License Code

If you have installed the U/SQL Manager, as part of the U/SQL Client software installation, then the **License Form** will also include a **License Upgrade Code** for it. In this case, after installing the U/SQL Client license for the ODBC Driver and Administrator, as described in the section [Entering a License Code](#), click the **Upgrade** button on the License Tool as directed by the License Installation Instructions.

You can also upgrade a license at any time, for example, to change from read-only to read-write or to change an evaluation license into a full license. To change a license, you will be supplied with a **License Form** containing the appropriate **License Upgrade Code**.

Whether you are upgrading the license during the U/SQL Client software installation to include the U/SQL Manager, or subsequently for any reason, then from the **License Tool** click the **Upgrade** button. The **License Code Upgrade** dialog box is displayed:



A sample Upgrade License Code is **S43A-UG1-F5-1-C19599**, where:

| | |
|--------|--|
| S43A | Same unique serial code of the original license. |
| UG1 | The number of the upgrade code (UG). In this case it is the first upgrade (number 1). |
| F5 | The upgraded features code, in this case licensing the U/SQL Manager to operate. |
| 1 | The copy number of this upgrade. You may have more than one user to upgrade. |
| C19599 | The checksum that checks that all parts of the License Upgrade Code are valid. Note, the checksum can be an alphanumeric code, for example, CB4XG3DUY. |

Enter the **License Upgrade Code** from the **License Form** and click **Apply**. If an incorrect upgrade code is entered an error message is displayed. You can make a correction to install a valid code or click **Close** to reject the upgrade.

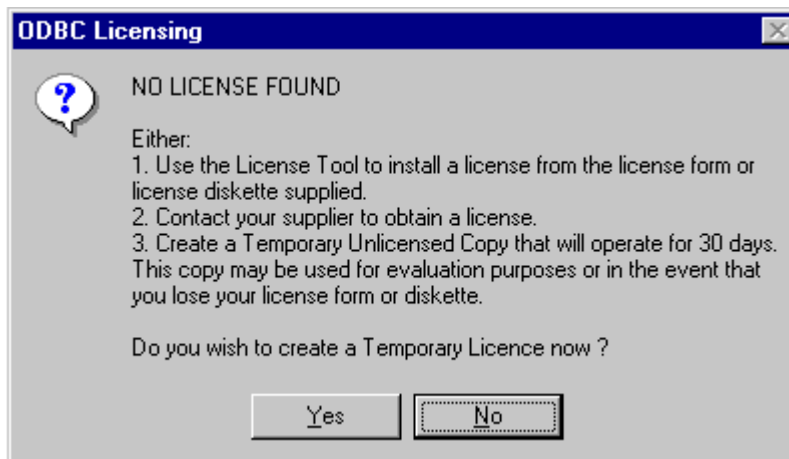
Creating a Temporary Unlicensed Copy

The U/SQL Client software may be operated as a **Temporary Unlicensed Copy** for 30 days, in unrestricted mode, without installing a license. This applies:

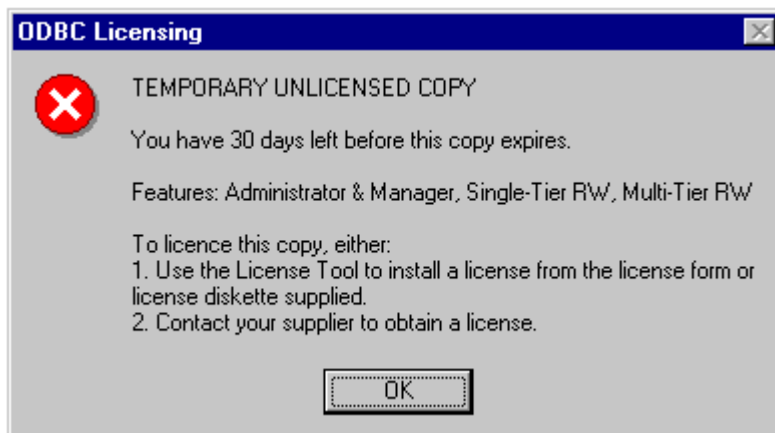
- If you remove the existing full license
- In the event of a disk crash, when after re-installing the U/SQL Client software, the **License Form** may not be immediately 'to hand'.

You can use the [License Tool](#) at any time to install a valid license, as described above.

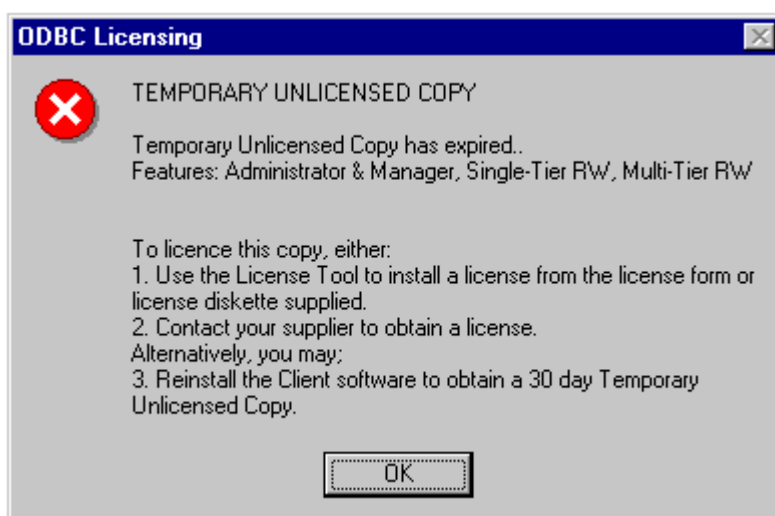
If no U/SQL Client license has been installed and an ODBC-enabled product, for example, Access, IQ or Impromptu, is invoked then the following message is displayed:



If you click **Yes** to create a **Temporary Unlicensed Copy**, then each time an ODBC-enabled product is invoked the following message is displayed:



After the **Temporary Unlicensed Copy** expires, the following message will be displayed:



Examining and Removing the U/SQL Client License

Use the **Examine** option of the [License Tool](#) to review the status of the installed license; be it a full license, a licensed copy with an expiry date or a Temporary Unlicensed Copy.

Use the **Remove** option of the **License Tool** to delete an existing evaluation or full license. Once a Temporary Unlicensed Copy has been created it cannot be removed except by installing a valid license, as described above.

Installing the U/SQL Server License

The U/SQL Server requires a license file to be created before it can operate. The license contains various elements including: the maximum number of concurrent U/SQL Client users that can connect to the server, the data source driver (say, C-ISAM), the server platform (say, IBM AIX), and, for an evaluation, the expiry date.

Note: *Each user is allowed 3 simultaneous connections from his/her PC and they count as ONE server user number (user #). If a user has more than 3 connections from the same PC, then a further server user number will be used.*

Each UNIX Interactive U/SQL utility, [usqli](#), connection counts as ONE server user number.

There are two options for creating a license for the U/SQL Server to operate:

1. If you have been issued with a License Form (see the sample in [Appendix B](#)) containing details of your **U/SQL Server License Code**, you will install this code.

A typical License Code is **S43A-U4-D1-P2-F4-C1GHT4FTL** where:

| | |
|------------------|--|
| S43A | The unique serial code for each license which MUST match the serial code on each client PC. |
| U4 | The maximum concurrent number of U/SQL Client users allowed to connect, in this case 4. |
| D1 | The Data Source Driver code, for example, Micro Focus COBOL. |
| P5 | The server platform code, for example, HP-UX. |
| F2 | The features code, determining Multiple-tier, read-only/read write and so on. |
| C1GHT4FTL | The checksum that check all the parts of the License Code and that the correct Organization name is entered. |

2. You can set up a 2-user Temporary License that will operate for 30 days.

Instructions for installing a U/SQL Server license for UNIX platforms and Windows NT Server are described in the following sections:

- [Installing the Server License on UNIX Platforms](#)
- [Installing the Server License on Windows NT Server.](#)

Installing the Server License on UNIX Platforms

On UNIX platforms, to install a license you execute the **serv_setup.sh** script, from the **bin** directory below the base directory of the U/SQL Server software installation, for instance, **/usr/usqls/bin**. It is invoked as follows:

```
cd /usr/usqls/bin
./serv_setup.sh
```

The Main Menu is displayed:

```
U/SQL SERVER CONFIGURATION AND CONTROL - MAIN MENU
```

```
=====
```

- 1) DATA SOURCE CONFIGURATION
- 2) CONFIGURATION SETTINGS
- 3) START SERVER
- 4) STOP SERVER
- 5) LOG FILE CONTROL
- 6) CREATE, VIEW & REMOVE SERVER LICENSE
- 7) VIEW THE 'usqlsd.ini' FILE

Enter '?' or '<NUMBER>?' for HELP, 'x' to EXIT.

Select option:

Select option **6**, **CREATE, VIEW & REMOVE SERVER LICENSE**, which displays the **Product Licensing Utility** menu:

```
Product Licensing Utility - Rev x.xx
```

Note: To create a Full License File you will need to have available the License Form containing the License Code

- 1 - Create Full License File
- 2 - Create Special License File (when upgrading to Rev 3)
- 3 - Create Temporary 2-User License File
- 4 - View License File
- 5 - View & Remove License in Memory
- 6 - Exit

Select option:

Now you can either create a full license, a special license (if you have upgraded to U/SQL Revision 3 or above) or a temporary license, as described in the sections below.

Creating a Full License

If you have been provided with a License Form, which includes details of your U/SQL Server License Code, then from the **Product Licensing** utility menu, select **1** to Create a Full License File. The following is displayed:

```
Create Full License File
```

You must enter your Organization Name and License Code EXACTLY as shown on your License Form.

```
Your Organization: ABC Company
License Code: S43A-U4-D1-P5-F2-C1GHT4FTL
```


If the Organization name or the License Code be incorrectly entered then you will be notified by the following message and you can either exit or re-enter the combination again:

This is not a valid License Code, please check your License Form and re-enter.

Enter R to retry or E to exit: **R**

If the Organization and License Code combination is correctly entered then the following is displayed:

This License has the following characteristics

Product: **U/SQL Adapters**

Serial code: **S43A**

Create date: **7 Jan, 1998**

Expiry date: **None**

Server platform: **HP-UX**

Features:

Multi-tier

Data source driver(s):

Micro Focus COBOL

Max usage counts:

U/SQL Client Users: 4

Please confirm that you agree to abide by the terms of the License Agreement.

I agree by these terms (Y/N): **Y**

If you answer 'N' then the following message is displayed:

The License File cannot be created without your agreement to the License terms.

Enter R to retry or E to exit:

If you answer 'Y' then the following is displayed:

The default path and License File name is: **/usr/usqls/bin/SERVER.LIC**

Please enter:

- a) the License File name (which must have a .LIC extension) to create it in the current directory
- b) the full path and the License File name, or
- c) NEWLINE to accept the displayed default

Enter:

Normally, you will simply accept the default license file name, **SERVER.LIC**, by pressing the RETURN key.

The following error conditions may be encountered:

- If a duplicate license file name exists, the following message is displayed and you can either replace the existing license with this new one or choose a new name as above.

A license file already exists with this pathname. Do you want to replace it (Y/N): **N**

- If you do not include a '.LIC' or '.lic' suffix with the license file you will be presented with the following message and you are requested to re-enter the name:

Transoft U/SQL User Guide

The license pathname must have a .LIC or .lic suffix and the filename portion must not be empty.

- If the path into which you wish to save the license file does not exist you will be presented with the following message and you are requested to re-enter the path:

No such directory.

If the license file is successfully created then you must set up the **'License='** (note spelling!) directive, to denote the path and name of the license file, in the UNIX **usqlsd.ini** configuration file in the **[Configuration Settings]** section.

License file has been successfully created!

Do you wish to update the usqlsd.ini configuration file with this License:

License=**/usr/usqls/license/SERVER.LIC**

Current setting : **/usr/usqls/bin/SERVER.LIC**

Enter (Y/N): **Y**

usqlsd.ini updated.

At this point the license installation is complete and you are returned to the [Product Licensing Utility](#) main menu.

Note: The **License=** directive can also be set using the **serv_setup.sh** script, and the **CONFIGURATION SETTINGS** option. You will be prompted to '**Enter license file**' name; enter the path and name.

Creating a Temporary 2-User License

For evaluations or in case you have mislaid your License Form, containing the U/SQL Server full License Code, you can create a temporary 2-user license that will operate for 30 days.

To create a temporary license select the **Create Temporary 2-User License File** option from the [Product Licensing Utility](#) main menu. The following is displayed:

Create Temporary 2-User License File

Your Organization: **ABC Company**

This license has the following characteristics

Product: **Any**

Serial code: **S999999999**

Create date: **7 Jan, 1998**

Expiry date: **6 Feb, 1998**

Server platform: **Any**

Max usage counts:

Users: **2**

This is a Temporary license.

Please confirm that you agree to abide by the terms of the License Agreement.

I agree by these terms (Y/N): **Y**

Then continue as in the section [Creating a Full License](#) to provide the license file name.

Display a License File

You can display the contents of a License file by selecting the **View License File** option from the [Product Licensing Utility](#) main menu. The following is displayed:

You have the following License file(s) in the current directory:

```
SERVER.LIC
ABC.LIC
```

Enter the (path and) name of the License file you wish to view, or press NEWLINE to return to the main menu:

```
SERVER.LIC
```

```
Organization: ABC Company Inc
License Code: S43A-U4-D1-P5-F2-C1GHT4FTL
```

This License has the following characteristics

```
Product: U/SQL Adapters
Serial code: S43A
Create date: 7 Jan, 1998
Expiry date: None
Server platform: HP-UX
Features:
    Multi-tier
Data source driver(s):
    Micro Focus COBOL
Max usage counts:
    U/SQL Client Users: 4
```

This is a Special or Temporary license (as appropriate)

Note: *Read-Write* is the default and is not explicitly shown as a feature.

View and Remove License in Memory

When the U/SQL Server is started with a license defined by the directive **License=**, in the **usqlsd.ini** configuration file, the license is set up as a memory table to hold the maximum number of concurrent U/SQL Client users.

You can view the contents of a license memory table by selecting the **View & Remove License in Memory** option from the **Product Licensing** Utility. This option also allows you to remove an individual user from the memory table if, for example, his/her PC has 'gone down' and lost the connection to the U/SQL Server. Optionally, all users have to be removed from the memory table before the license table itself is removed from memory altogether.

Note: *Great care must be exercised to ensure that ONLY users, whose PCs have become in-operable, are removed from the license memory table. Removing a user from a server user number removes ALL the user's connections in that server user number. If a user is removed with a valid current connection, then that user will cease to operate.*

Selecting the **View & Remove License in Memory** option from the **Product Licensing** utility displays:

Transoft U/SQL User Guide

View & Remove License in Memory

You have the following Licenses in memory:

- 1) /usr/usqls/bin/SERVER.LIC
- 2) /usr/usqls/bin/ABC.LIC

Enter a number in the range 1 to 2 to select license: 1

License file: /usr/usqls/bin/SERVER.LIC
Organization: ABC Company Inc
License string: S43A-U4-D1-P5-F2-C1GHT4FTL

This license has the following characteristics

Product: **U/SQL Adapters**
Serial code: **S43A**
Create date: **7 Jan, 1998**
Expiry date: **None**
Server platform: **HP-UX**
Features:
 Multi-tier
Data source drivers(s):
 Micro Focus COBOL
Max usage counts:
 U/SQL Client Users: 4

Do you wish to view the Active Users (Y/N)? **Y**

Active U/SQL Client Users:3 (max 3 connection per user#)

| User# | User IP | User Name | User's Count |
|-------|-------------|-----------------------------------|--------------|
| 1 | 128.1.10.13 | Joe Smith | 1 |
| | | Server PID(s) : 28883 | |
| 2 | 128.1.10.20 | Jim Jones | 3 |
| | | Server PID(s) : 28804 28824 28851 | |
| 3 | 128.1.10.20 | Jim Jones | 1 |
| | | Server PID(s) : 28943 | |

Enter User# or 'A' for All to remove user(s), or press NEWLINE to return to the main menu: 1

User(s) removed!

Enter User# or 'A' for All to remove user(s), or press NEWLINE to return to the main menu:

If there are no active users then the following is displayed:

There are no Active Users!
Do you wish remove the license from memory (Y/N): **Y**

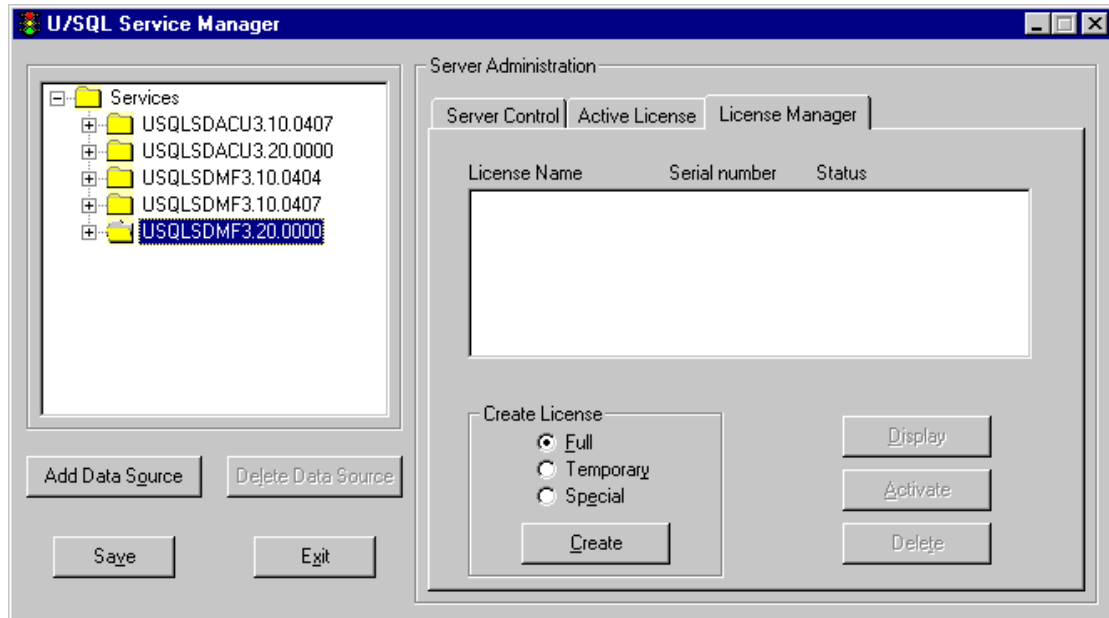
License removed!

Press NEWLINE to return to the main menu.

You will notice that each user is allowed 3 simultaneous connections from his/her PC and they count as one server user number (user #). If a user, Jim Jones in the example above, have more than 3 connections from the same PC, then a further user # will be used.

Installing the Server License on Windows NT Server

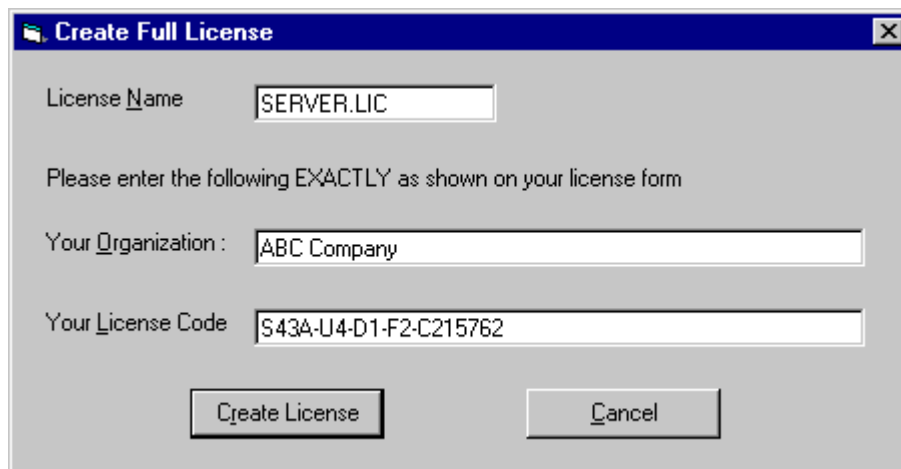
On Windows NT Server, the U/SQL Server license is installed via the [U/SQL Service Manager](#) utility contained in the U/SQL program group. Run the program and from the main property sheet, select your service, for example, **USQLSDMF3.20.0000**. Click the **License Manager** tab. The **License Manager** property page is displayed:



From the **Create License** section you have the option of creating a **Full**, **Temporary** or **Special** license.

Creating a Full License

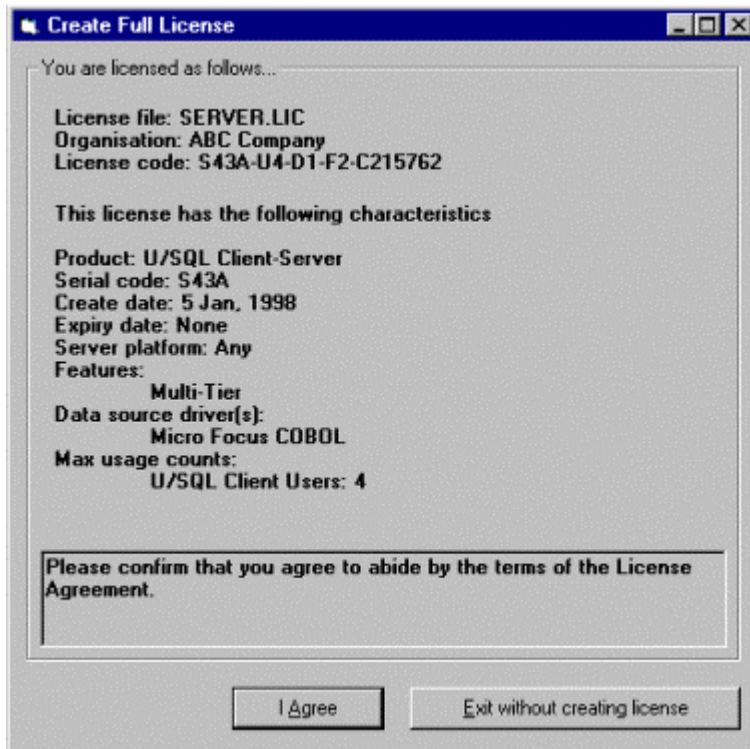
If you have been provided with a **License Form** which includes details of your U/SQL Server License Code, select the **Full** option to create a Full license and then click the **Create** button. The **Create Full License** dialog box is displayed:



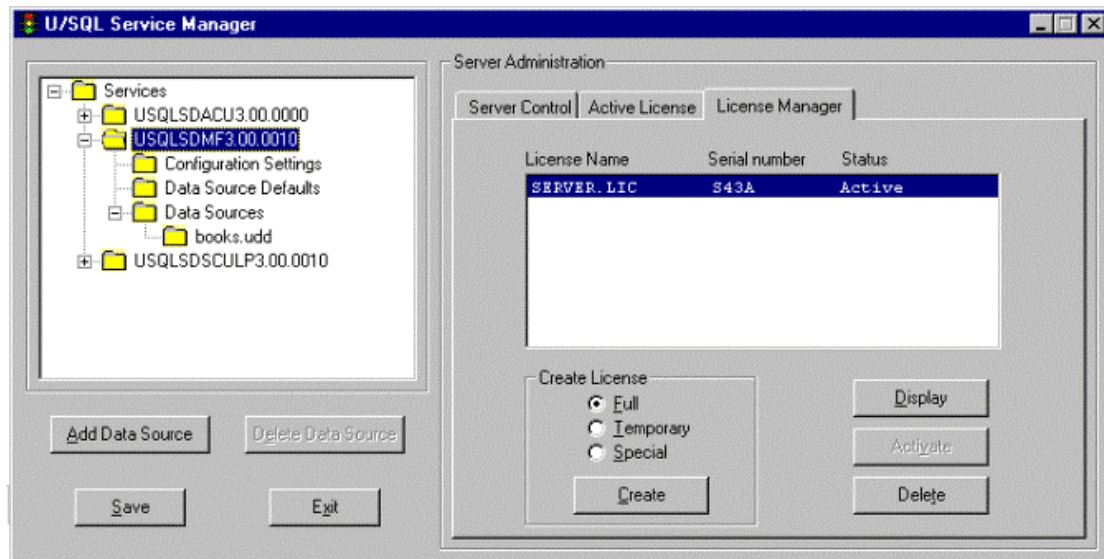
Enter **Your Organization** and **Your License Code** EXACTLY as shown on your License Form. The default **License Name** of **SERVER.LIC** can be changed, but it must have a '.LIC' extension. Then click the **Create License** button.

Note: The checksum can be an alphanumeric code, for example, C1GHT4FTL.

If an incorrect Organization/ License Code combination is entered an error message is displayed. You can make a correction to install a valid combination or click **Cancel**. Assuming a valid combination is entered, then the following dialog box is displayed:



If you agree to abide by the terms of the License Agreement then click the **I Agree** button. The License Manager is displayed showing the new license:



To activate a license you highlight any License Name and click the **Activate** button. Usually you will only have one license.

By making a license active you are automatically setting up the **License** directive in the Registry as the License Name, for example:

License=SERVER.LIC

Creating a Temporary 2-User License

For evaluations or if you have mislaid your License Form, containing the U/SQL Server full License Code, you can create a temporary 2-user license that will operate for 30 days.

To create a temporary license, select the **Create License\Temporary** option from the **License Manager** property page of the [U/SQL Service Manager](#), and click the **Create** button. The **Create Temporary License** dialog box is displayed:

Enter **Your Organization** name and, if necessary, change the default **License Name** of SERVER.LIC. Then continue, as described in the [Creating a Full License](#) section above, by agreeing to the License Agreement before activating the license.

Display a License

You can display the contents of any license by highlighting the required **License Name** and clicking the **Display** button on the **License Manager** property page of the U/SQL Service Manager. A form showing the **License Details** is displayed:

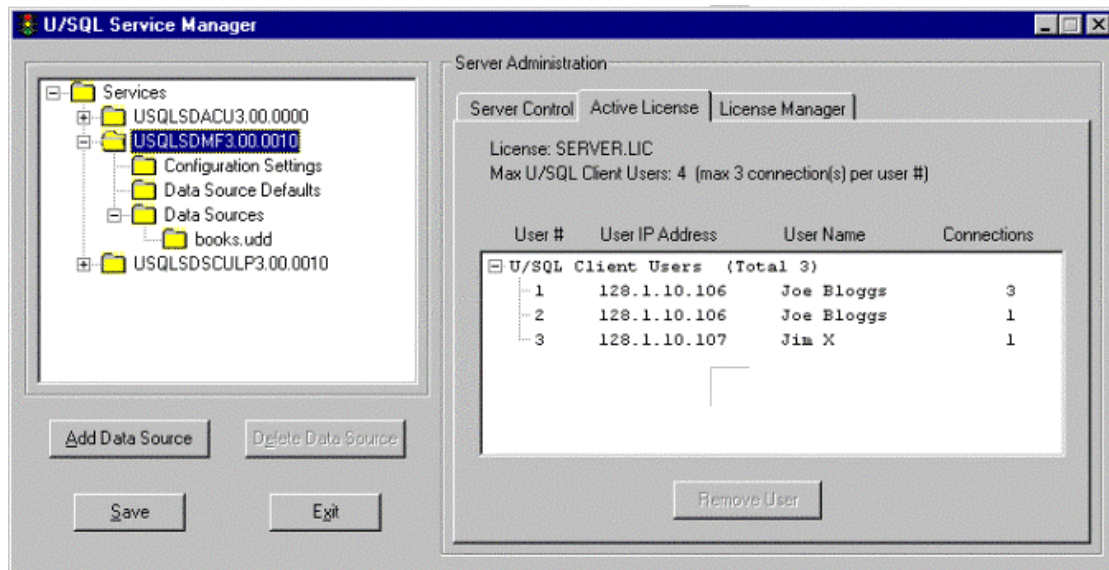
Note: *Read-Write* is the default and is not explicitly shown as a feature.

Also, the **License code** checksum can be an alphanumeric code, for example, C1GHT4FTL.

View & Remove License in Memory

When the U/SQL Server is started with a particular license, defined by the **License=** directive, in the Registry, the license is set up as a memory table to hold the maximum number of concurrent U/SQL Client users that have been licensed.

You can view the contents of a license memory table by selecting the **Active License** tab on the U/SQL Service Manager. The **Active License** property page is displayed:



This option also allows you to remove an individual user from the memory table if, for example, his/her PC has 'crashed' and lost the connection to the U/SQL Server. Highlight the user to be removed and click the **Remove User** button.

Note: Great care must be exercised to ensure that **ONLY** users, whose PCs have become in-operable, are removed from the license memory table. Removing a user from a server user number (user #) removes ALL the user's connections in that user #. If a user is removed with a valid current connection, then that user will cease to operate.

Each user is allowed 3 simultaneous connections from his/her PC and they count as one server user number (user #). If a user has more than 3 connections from the same PC, then a further user # will be used.

To remove an active license memory table the U/SQL Server must first be stopped from the **Server Control** property page of the [U/SQL Service Manager](#). If there are no active users then on the Active License form you will be presented with a **Reset License Table** button, which when clicked will clear the memory table for the license.

Starting U/SQL with Licensing

After you have installed your U/SQL Client and, for Multiple-tier, U/SQL Server software and licenses as described in the sections above, you can start to use U/SQL. The following section describe points to be careful of, possible error conditions that may occur, the reasons for them and the remedies.

Starting the U/SQL Server with the License File

For Multiple-tier versions of U/SQL, you will have created a U/SQL Server license file, as described in the section Installing the U/SQL Server License above, as either a full license, a special license (if you are upgrading your U/SQL software from a revision prior to Revision 3 and you are continuing to use the existing client token licensing) or as a temporary 2-user license.

Ensure that the path and name of your license file are defined correctly by the U/SQL Server **License=** directive (note spelling!), in the UNIX **usqlsd.ini** configuration file or Windows NT Server Registry, in the **[Configuration Settings]** section, otherwise the U/SQL Server will not find the license file when starting.

Example entries on UNIX and Windows NT Server are:

```
License=/usr/usql/bin/NEWLIC.LIC    UNIX
```

```
License=NEWLIC.LIC                Windows NT
```

When you start the U/SQL Server, either on a UNIX or Windows NT Server platform the U/SQL Server will attempt to set up the license memory table defined by the **License=** directive.

The U/SQL Server may fail to start, for example under UNIX, as follows:

```
U/SQL Server [Engine 3.00.0000]
Copyright © Transoft Ltd 1994-98

Server has not started due to:
Unable to access license file: NEWLIC.LIC

Server stopped.
```

Possible error conditions that may be encountered are:

- (E3000) Unable to access .ini file: usqlsd.ini
For UNIX installations, ensure that the **usqlsd.ini** configuration file is in the same directory where the U/SQL Server is started, usually **/usr/usqls/bin**.
- (E3001) Unable to find directive (License=)
Ensure that this directive is set up in the UNIX **usqlsd.ini** file or the Windows NT Server Registry.
- (E3002) Unable to access license file: <license.lic_file>

Ensure the license path and file name agree with the **License=** directive in the UNIX **usqlsd.ini** file or the Windows NT Server Registry.

- (E3003) Unable to locate any license files in current directory: <path>

Ensure that either the license file is in the current directory where the U/SQL Server is started, or the license path and file name agree with the **License=** directive in the UNIX **usqlsd.ini** file or the Windows NT Server Registry.

- (E3004) <license.lic_file> is not a valid license file

Either ensure the license path and file name agree with the **License=** directive in the UNIX **usqlsd.ini** file or the Windows NT Server Registry, or re-create the license file, as it may have become corrupt.

- (E3005) File <license.lic_file> contains an invalid license code

Either ensure the license path and file name agree with the **License=** directive in the UNIX **usqlsd.ini** file or the Windows NT Server Registry, or re-create the license file, as it may have become corrupt.

- (E3006) License file <license.lic_file> contains an expired license code

Either create a temporary 2-user license or contact your supplier to obtain a full license.

- (E3007) License file <license.lic_file> does not apply to the current platform

You have installed a license that cannot be used with the current platform. Either create a temporary 2-user license or contact your supplier to obtain a license with the correct platform.

- (E3008) License file <license.lic_file> does not apply to this product

This is not a license for U/SQL. Either create a temporary 2-user license or contact your supplier to obtain a license with the correct platform.

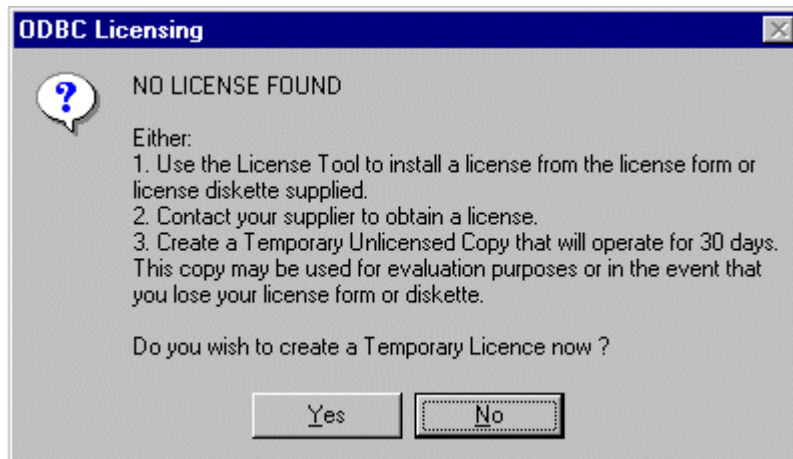
- (E3016) Incompatible Licensing module! Contact your supplier.

You have incompatible U/SQL Server software, contact your supplier.

Assuming the U/SQL Server finds the license file and it is valid then, after displaying the license details, it will start.

Starting the U/SQL Client with Licensing

The U/SQL Client software operates with the client license. If no license is present, the existing license is found to be corrupt or a non-temporary unlicensed copy has expired, then you will be offered the following choices:



You then make your choice. Refer to the section [Installing the U/SQL Client License](#) for details of the various options.

For Multiple-tier, once the U/SQL Client has verified the client license as valid, it will attempt to connect to the U/SQL Server. If it does not connect successfully it may be due to one of the following conditions:

- **Incompatible Client and Server Licensing!**

Client software Serial Code: S12345A. Server software Serial Code: S1111A

Either:

1. Install compatible Licenses on Client and Server.
2. Use the License Tool on the Client Software to create a Temporary Unlicensed Copy that will operate for 30 days.
3. Contact your supplier to obtain a full license.

- **Incompatible Temporary Server License!**

Either more than one Data Source or an RDBMS Data Source included in Server software.

DSD(s) in Server: ACUCOBOL, C-ISAM

Contact your supplier to obtain a full license.

A temporary 2-user U/SQL Server license will only work with a single data source driver and no RDBMS data sources.

- **Server License Expired!**

Server License Serial Code: S99999A. Expiry date: DD-MMM-YYYY

Either:

- Create a 30 day 2-user Temporary License for the Server
- Contact your supplier to obtain a full license.

- **Server License - incompatible Data Source(s)!**

In Server: **ACUCOBOL**. In License: **Micro Focus**. Server License Serial Code: **S12345A**

Contact your supplier to obtain a full license.

In the meanwhile you can create a temporary 2-user license.

- Licensing Problem!

<Error message>

Contact your supplier.

Where <Error message> may be one of the following:

- **(E3015) Duplicate client license detected**

The same U/SQL Client License Code has been installed on more than ONE PC and is trying to connect. Install a unique license code on each PC.

- **(E3011) The maximum licensed user count has been reached**

You are at the limit of your concurrent Server license connections. Obtain an increased connection count license from your supplier.

- **(E3012) The license has expired**

Either create a temporary 2-user U/SQL Server license or contact your supplier to obtain a full license.

- **(E3013) Warning: <days> days remain on this license**

This U/SQL Server license will expired in <days>.

- **(E3009) Invalid license key**

The license the U/SQL Server originally connected to is no longer present. It may have been removed by using the **serv_setup.sh** script. This will happen on starting an U/SQL Client.

- **E3010 Invalid user type code**

An invalid client user type has attempted to connect.

- **E3014 Unknown user ID**

This error message will appear, in the **usqlcs.log** file, if a user is removed from the license memory using **serv_setup.sh** when he/she is still connected. No error can be returned to the client.

Unattended Client installation

Unattended Client installation

Introduction

This document describes the necessary steps required to install the U/SQL Client application in an "unattended" or "silent" mode, suitable for rolling out to multiple client machines automatically. An unattended install prompts for no user interaction, the decisions required to perform the installation process having previously been specified in a response file.

How to perform Unattended (Silent) installations

Initial setup

Copy the contents of the <cdrom>\32bit\Disc1 directory to a directory on a share on the file server machine from which you will be using the unattended installation, for instance \\SVRNAME\SHARENAME\USQLINST.

Note: This document uses the shorthand "USQLINST" to refer to the nominal "\\SVRNAME\SHARENAME\USQLINST" path.

Generating a response file - Overview

A response file is a text file that is generated from user interaction when running a setup program. A response file contains details of the buttons pressed, the options changed and the text typed in the setup process.

When creating a response file, you cannot specify its location or name, it will always be created in the %WINROOT% folder (eg "C:\WINDOWS" or similar), and will always be called "SETUP.ISS".

Response files can be generated by running the setup program in "Record" mode and installing the software on a reference machine, choosing the setup options you will later use for the other client workstations.

This response file can then be used to rollout the application on multiple machines so that the Setup program does not prompt for user interaction, but instead selects options from the response file.

Example response files have been included in the <cdrom>\32bit\Disc1\UNATTEND directory, and can be viewed using any text viewer or editor.

The example files are:

| File name | Description |
|--------------------|--|
| COMPACT.ISS | "Compact" setup type chosen, "Multi-tier ODBC Client" password entered, default paths and options accepted throughout the rest of the installation |

| | |
|--------------------|---|
| | process. |
| TYPICAL.ISS | "Typical" setup type chosen, "Multi-tier ODBC Client" password entered, default paths and options accepted throughout the rest of the installation process. |

Generating a response file – Steps

To generate a response file:

1. From the "UNATTEND" directory within the USQLINST directory, run GenResp.bat. This will start the setup.exe program in "Record mode", ready to record the desired user input.
2. Input responses to the install program's prompts as you would want them to be answered on other client machines.
3. The setup program will record the actions you take performing the setup process in a "response file".
4. When the install process is complete, move the generated response file (SETUP.ISS) from the %windir% directory, (often C:\WINDOWS or C:\WINNT), to a location like USQLINST\UNATTEND.

You can then rename the response file, or modify it with a text editor if required.

Licensing

The U/SQL application has specific licensing requirements, and the interactive client installation displays the client licence manager so that the user can input licence details by hand.

For the unattended installation, the licence information can instead be specified in the SETUP.INI file in the main USQLINST directory.

To do this:

1. Open the SETUP.INI file in the main USQLINST directory using a text editor

The file will look something like this:

```
[Startup]
AppName=U/SQL Client Installation
FreeDiskSpace=684
[Registry]
UKSupportEmail=UKSupport@transoft.com
USSupportEmail=USSupport@transoft.com
[Configuration]
Options=51
Features=6
```

2. Add a new section to the bottom of the SETUP.INI file, the section should look like this:

```
[Licence]
Key=XXXXX-XX-XXX-X-XXXXXXXXXX
Name=YourNameHere
Organisation=YourOrganisationHere
```

Where:

“Key” is the licence key you would usually use when installing client machines

“YourNameHere” is the Name as you would enter it into the licence manager.

“YourOrganisationHere” is the Organisation as you would enter it into the licence manager.

3. Save the SETUP.INI file and exit.

The silent installation process will not fail if a licence is not specified in the SETUP.INI file.

If a client is installed with a licence specified in the SETUP.INI file, you can do one of two things:

1. Enter the licencing information into the SETUP.INI and run the silent setup process again.
2. Enter the licence key by hand using the Client Licence Manager.

Running an unattended installation

Once a response file has been generated, an unattended installation can be run on the target workstations:

“Silenttypical.bat” and “Silentcompact.bat” are examples of batch files used to run unattended installations are included in the UNATTEND directory within the main USQLINST directory.

The “Silenttypical.bat” file looks like this:

```
..\setup -s -f1.\typical.iss -f2.\USQLtypical.log
```

This runs the setup.exe program from the USQLINST directory, using the “typical.iss” response file, and generates a log file called “USQLtypical.log”.

The batch files provided are only an example, and should be copied and modified as required.

Checking the Installation log file

If a silent installation is suspected to have completed abnormally, you should check the installation log file. If the installation was started from the “Silenttypical.bat” or “Silentcompact.bat” example batch files, the installation log file will be called “USQLtypical.log” or “USQLcompact.log” respectively, and is created in the “UNATTEND” directory by default.

On successful completion of an installation, the log file will look something like this:

```
[InstallShield Silent]
Version=v7.00
```

File=Log File

[ResponseResult]

ResultCode=0

"ResultCode=0" indicates that the install process finished successfully.

If the ResultCode is NOT 0, then the installation has failed, - check Appendix A ("InstallShield Common Errors") for more information.

Note: Installing the U/SQL Client requires Administrative rights on the client workstation.

Note: Attempting to install the U/SQL Client in Silent Mode without Administrative rights will fail, often with a result code of "-1" or "-11".

InstallShield - Common error codes

Common InstallShield errors that may be seen on the ResultCode line of the ".LOG" file generated when running the unattended (silent) installation.

| Result Code | Description |
|--------------------|---|
| 0 | Install completed successfully (Not an error). |
| -1 | General error |
| -2 | Invalid mode |
| -3 | Required data not found in the Setup.iss file. (The response file is not correct for the installation being performed). |
| -4 | Not enough memory available. |
| -5 | File does not exist. |
| -6 | Cannot write to the response file. |
| -7 | Cannot write to the log file. |
| -8 | Invalid path to the InstallShield Silent response file. |
| -9 | Not a valid list type (string or number). |
| -10 | Data type is invalid. |
| -11 | Unknown error during setup. |
| -12 | Dialog boxes are out of order. |
| -51 | Cannot create the specified folder. |
| -52 | Cannot access the specified file or folder. |
| -53 | Invalid option selected. |
| -5001 | Generic error |
| -5002 | Failed reading media header |
| -5003 | Failed installing kernel |
| -5004 | Failed starting kernel |
| -5005 | Failed opening CAB |
| -5006 | Failed installing support |
| -5007 | Failed setting text substitution |

| | |
|-------|--|
| -5008 | Failed initializing setup info |
| -5009 | Failed getting setup driver |
| -5010 | Failed initializing properties |
| -5011 | Failed running setup driver |
| -5012 | Failed uninstalling support |
| -5013 | Failed to extract file from setup boot file |
| -5014 | Failed to download file [occurs only when saving setup files during an Internet setup] |
| -6001 | Failed starting the setup launcher |
| -6002 | Failed finding the setup launcher |
| -6003 | Failed loading the setup launcher |
| -6004 | Failed verifying the signature of setup launcher |
| -6005 | Failed installing the setup launcher to proper location |
| -6006 | Failed extracting setup launcher |

Configure and Use

ODBC Overview

ODBC Overview

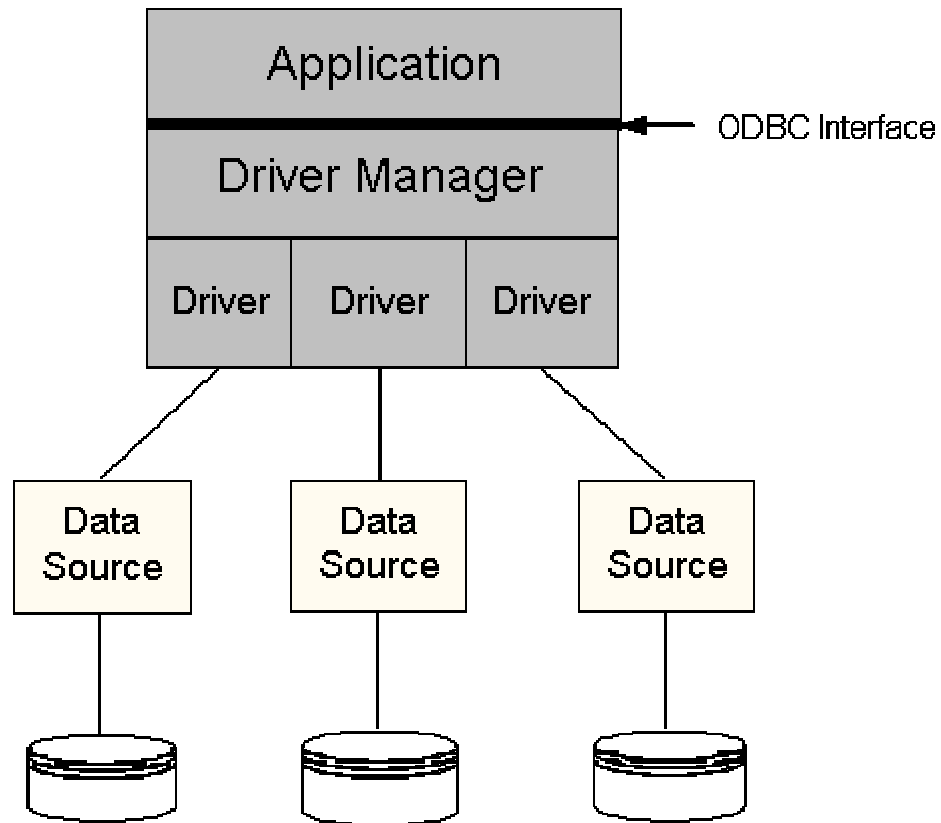
Microsoft's **Open DataBase Connectivity (ODBC)** standard is the most popular means for connecting Windows products and applications to relational database servers. Using Transoft's U/SQL product, you can connect ODBC Windows products, such as Microsoft Visual Basic, Microsoft Access, and Powersoft PowerBuilder, to your non-relational files, allowing you to both report on and update the data.

ODBC Components

The ODBC architecture has four components:

- **Application** - Performs processing and calls ODBC functions to submit SQL statements and retrieve results.
- **Driver Manager** - Loads drivers on behalf of an application.
- **Driver** - Processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS.
- **Data source** - Consists of the data the user wants to access and its associated operating system, DBMS, and network platform (if any) used to access the DBMS.

The Driver Manager and driver appear to an application as one unit that processes ODBC function calls. The following diagram shows the relationship between the four components:



U/SQL is available in two models:

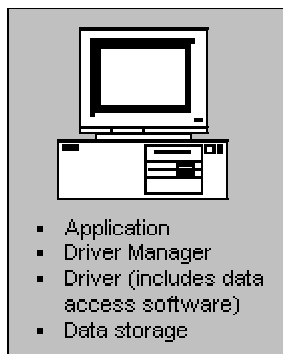
- [Single-tier](#)
- [Multiple-tier](#).

The Single-tier model of U/SQL

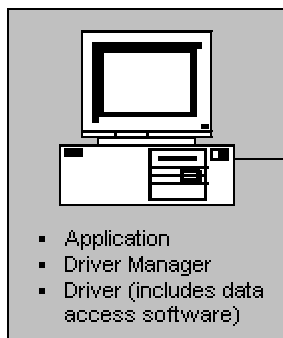
In a single-tier implementation, the database file is processed directly by the driver. The driver processes SQL statements and retrieves information from the database. A driver that manipulates an Xbase file is an example of a single-tier implementation.

The diagram below shows two types of Single-tier configurations.

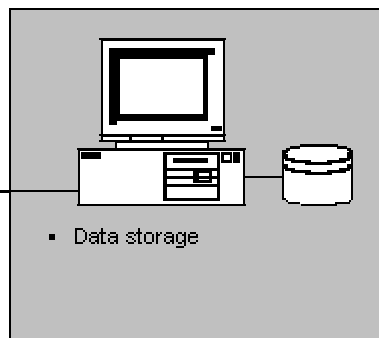
System A



Client B



Server B



In the Single-tier model of U/SQL, both the Client and Server elements of the system reside on the same platform, for example, a Windows PC.

The Multiple-tier model of U/SQL

In a Multiple-tier configuration, the driver sends SQL requests to a server that processes SQL requests.

Although the entire installation may reside on a single system, it is more often divided across platforms. The application, driver, and Driver Manager reside on one system, called the client. The database and software that control access to the database typically reside on another system, called the server.

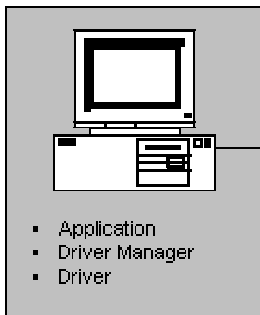
Another type of Multiple-tier configuration is a gateway architecture. The driver passes SQL requests to a gateway process, which in turn sends requests to the data source.

The following diagram shows three types of Multiple-tier configurations. From an application's perspective, all three configurations are identical.

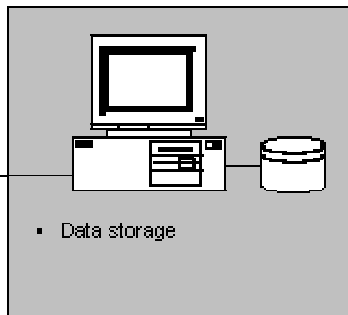
System C



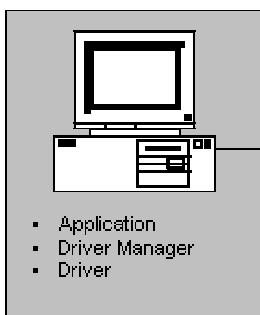
Client D



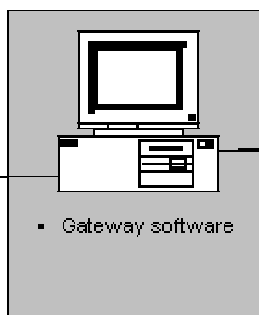
Server D



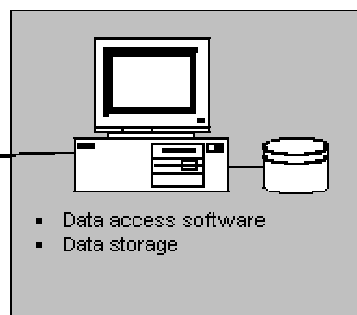
Client E



Server E1



Server E2



In the Multiple-tier model of U/SQL, the U/SQL ODBC driver resides on the Client platform, which is connected via a LAN to the Server platform, where the U/SQL Server and data source driver reside. For example, your Client platform could be a Windows PC and your Server platform a UNIX or Windows NT machine.

ODBC Compliance

The Microsoft Open Database Connectivity (ODBC) standard defines two areas of conformance:

- ODBC API, which specifies the functions that are supported
- ODBC SQL Grammar and ODBC SQL Data Types.

In order for a supplier to claim that their driver conforms to a given API or SQL conformance level, it must support all the functionality in that conformance level. By convention ODBC aware products are classified using the following ODBC API conformance levels:

- **Core**
- **Level 1**
- **Level 2**

Each of these levels specifies an additional set of functionality to the previous level. This section sets out the requirements of each of the levels, and specifies how the U/SQL product conforms to them.

A middleware driver must at minimum conform to the requirements of the Core level for it to be considered as ODBC compliant. Although an application may have the capability of using specific functionality which a given driver does not support, it will not use that feature with the given driver. A driver can implement additional functionality to that required for conformance to a given level. An ODBC-enabled application can determine the functionality supported by a driver by using the ODBC calls, **SQLGetInfo**, **SQLGetFunctions** and **SQLGetTypeInfo**.

This section covers the following topics:

- [ODBC API conformance](#)
- [SQL conformance](#)
- [API coverage](#)
- [SQL grammar coverage](#)

ODBC API conformance

API Conformance is classified at three levels:

- **Core API**
- **Level 1 API**
- **Level 2 API**

It is recommended that drivers must support all the Level 1 API features.

SQL conformance

SQL Conformance is also classified at three levels:

- Minimum SQL Grammar
- Core SQL Grammar
- Extended SQL Grammar

Core SQL Grammar corresponds roughly to the X/Open and SAG SQL CAE specification (1992).

Transoft U/SQL is conformant with ODBC Revision 2 (with Revision 3 support for System Data Source Names), and complies with the ODBC API and SQL Grammar specified in the following tables:

API coverage

| Function | Level | U/SQL | Notes (see below) |
|----------------------------|--------------|--------------|-----------------------------|
| SQLAllocConnect | Core | YES | |
| SQLAllocEnv | Core | YES | |
| SQLAllocStmt | Core | YES | |
| SQLBindCol | Core | YES | |
| SQLBindParameter | Level 1 | PARTIAL | 1. |
| SQLBrowseConnect | Level 2 | NO | |
| SQLCancel | Core | PARTIAL | 2. |
| SQLColAttributes | Core | | 3. |
| SQLColumnPrivileges | Level 2 | NO | |
| SQLColumns | Level 1 | PARTIAL | 4. |
| SQLConnect | Core | YES | |
| SQLDataSources | Level 2 | | |
| SQLDescribeCol | Core | YES | |
| SQLDescribeParam | | NO | |
| SQLDisconnect | Core | YES | |
| SQLDriverConnect | Level 1 | YES | |
| SQLDrivers | Level 2 | DM | |
| SQLError | Core | YES | |
| SQLExecDirect | Core | YES | |
| SQLExecute | Core | YES | |
| SQLExtendedFetch | Level 2 | NO | 5. |

| | | | |
|----------------------------|------------|-----|--|
| SQLFetch | Core | YES | |
| SQLForeignKeys | Level 2 | NO | |
| SQLFreeConnect | Core | YES | |
| SQLFreeEnv | Core | YES | |
| SQLFreeStmt | Core | | |
| SQLGetConnectOption | Level 1 | YES | |
| SQLGetCursorName | Core | YES | |
| SQLGetData | Level 1 | YES | |
| SQLGetFunctions | Level 1 | DM | |
| SQLGetInfo | Level 1 | YES | |
| SQLGetStmtOption | Level 1 | YES | |
| SQLGetTypeInfo | Level 1 | YES | |
| SQLMoreResults | Level 2 | NO | |
| SQLNativeSql | Level 2 | NO | |
| SQLNumParams | Level 2 | YES | |
| SQLNumResultCols | Core | | |
| SQLParamData | Level 1 | YES | |
| SQLParamOptions | Level 2 | NO | |
| SQLPrepare | Core | YES | |
| SQLPrimaryKeys | Level 2 | NO | |
| SQLProcedureColumns | Level 2 | NO | |
| SQLProcedures | Level 2 | NO | |
| SQLPutData | Level 1 | YES | |
| SQLRowCount | Core | YES | |
| SQLSetConnectOption | Level 1 | YES | |
| SQLSetCursorName | Core | YES | |
| SQLSetParam | Deprecated | YES | |
| SQLSetPos | Level 2 | NO | |
| SQLSetScrollOptions | Level 2 | NO | |

| | | | |
|---------------------------|---------|-----|--|
| SQLSetStmtOption | Level 1 | YES | |
| SQLSpecialColumns | Level 1 | YES | |
| SQLStatistics | Level 1 | YES | |
| SQLTablePrivileges | Level 2 | NO | |
| SQLTables | Level 1 | YES | |
| SQLTransact | Core | YES | |

(DM = provided by the Driver Manager)

Notes:

1. **SQLBindParameter** supersedes **SQLSetParam** and includes support for INPUT/OUTPUT parameters as found in Stored Procedures which is not yet supported.
2. **SQLCancel** does not interrupt a currently-preparing query unless asynchronous execution is activated. For information on this contact Transoft.
3. Some additional attributes have been added in ODBC 2 which are not yet supported. An example is Column Label, which is used by report writers to give columns different printed names.
4. 'Qualified Table Names' are not generally supported.
5. Only single-row result sets are supported.

SQL grammar coverage

| Statement / Element | Level | U/SQL | Notes (see below) |
|----------------------|----------|---------|----------------------|
| ALTER TABLE | Core | PARTIAL | 1. |
| ALTER TABLE DROP | Extended | NO | |
| CREATE INDEX | Core | PARTIAL | 2. |
| CREATE TABLE | Minimum | PARTIAL | 3. |
| CREATE VIEW | Core | YES | |
| DELETE WHERE CURRENT | Extended | YES | |
| DELETE | Minimum | YES | |
| DROP INDEX | Core | PARTIAL | 3. |
| DROP TABLE | Minimum | PARTIAL | 3. |

| | | | |
|-----------------------------------|----------|---------|--|
| DROP VIEW | Core | YES | |
| GRANT | Core | NO | |
| INSERT INTO VALUES | Minimum | YES | |
| INSERT INTO SELECT | Core | YES | |
| ODBC Procedure Extension | Extended | PARTIAL | |
| REVOKE | Core | NO | |
| SELECT | Minimum | YES | |
| SELECT .. GROUP | Core | YES | |
| SELECT .. UNION | Extended | YES | |
| SELECT .. FOR UPDATE | Extended | YES | |
| statement ; statement ; .. etc .. | Extended | NO | |
| UPDATE .. WHERE CURRENT | Extended | YES | |
| UPDATE | Minimum | YES | |
| AVG, MAX, MIN, SUM | Core | YES | |
| scientific notation nnE+mm | Core | NO | |
| FLOAT, DOUBLE PRECISION, REAL | Core | YES | |
| qualified objects | Core | NO | |
| BETWEEN | Core | YES | |
| binary-literal | Extended | NO | |
| BINARY, VARBINARY | Extended | NO | |
| bit-literal | Extended | NO | |
| BIT | Extended | NO | |
| NOT, AND, OR | Minimum | YES | |
| CHAR, VARCHAR | Minimum | YES | |
| LONG VARCHAR | Minimum | NO | |
| Comparison Predicate | Minimum | YES | |

| | | | |
|---------------------------------|----------|-----|--|
| DATE | Extended | YES | |
| AVG, COUNT, ..(DISTINCT) | Core | YES | |
| Dynamic Parameter (?) | Minimum | YES | |
| DECIMAL, NUMERIC, INTEGER | Core | YES | |
| BIGINT | Extended | NO | |
| EXISTS Predicate | Core | YES | |
| IN Predicate | Core | YES | |
| LIKE Predicate | Core | YES | |
| LIKE Predicate with ODBC escape | Extended | YES | |
| ODBC Date Literal | Extended | YES | |
| NULL Predicate | Minimum | YES | |
| ODBC Time Literal | Extended | YES | |
| ODBC Timestamp literal | Extended | YES | |
| ORDER BY | Minimum | YES | |
| ODBC Outer Join | Extended | YES | |
| ODBC Scalar Functions | Extended | YES | |
| quantified predicate | Core | YES | |
| AS alias | Core | YES | |
| TIME, TIMESTAMP | Extended | YES | |

Notes:

1. *Applies only to UDD tables.*
2. *Applies only to UDD tables. Does not build an index for a table already containing data rows.*
3. *Applies only to UDD.*

System and User Data Sources

If Microsoft's ODBC Driver Manager 3.0 or later is detected on your PC, then a **System DSN** tab in addition to a **(User) Data Sources** tab are included on your [U/SQL Administrator](#) property sheet, otherwise only a single Data Sources tab is provided.

Some newer 32-bit applications look for System Data Sources (requiring Microsoft's ODBC Driver Manager 3.0 or later to be installed) rather than User Data Sources. In general, if a System data source has been setup any 32-bit application used by any user should be able to use this data source. However, you may need to set up both System and User Data Sources for differing ODBC-enabled applications you are running.

A System Data Source is separately set up and configured in exactly the same way as a User Data Source.

File Data Sources

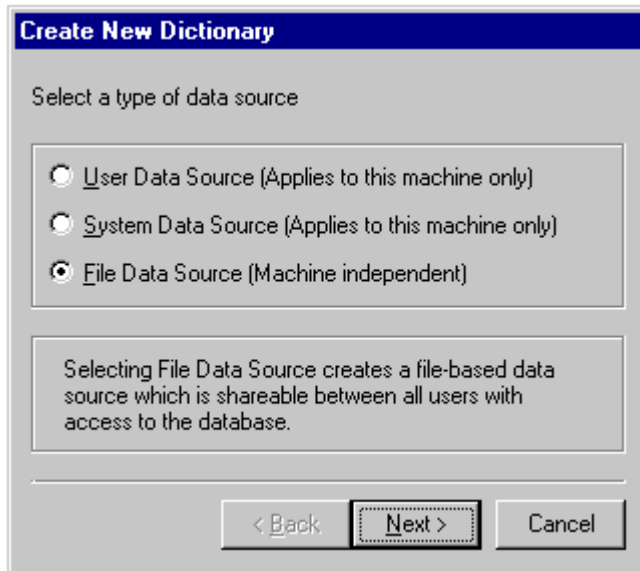
U/SQL 3.10.400 and above provide support for File DSNs. A File DSN is an ODBC File data source that stores connection information for a database in a text file with a '.dsn' extension. The connection information consists of parameters and corresponding values that the ODBC Driver Manager uses to establish a connection. File DSNs allow you to connect to a data provider, and can be shared by users who have the same drivers installed.

File DSNs can be created and maintained using:

- [U/SQL Manager](#)
- [U/SQL Administrator](#)
- [Win U/SQLi](#).

U/SQL Manager

Click on the **New Data Dictionary** button and the **Create New Dictionary** dialog box is displayed:



This dialog box contains the following options:

| | |
|---------------------------|---|
| User Data Source | The User Data Source option is selected by default. This creates a data source, which is specific to your machine, and visible only to you. |
| System Data Source | Selecting the System Data Source option creates a data source, which is specific to your machine, and usable by any user who logs onto your machine. |
| File Data Source | Selecting the File Data Source option creates a data source, which is independent of your machine. |

To create a File DSN, select the **File Data Source** option. The **Create New File Data Source** dialog box is displayed:

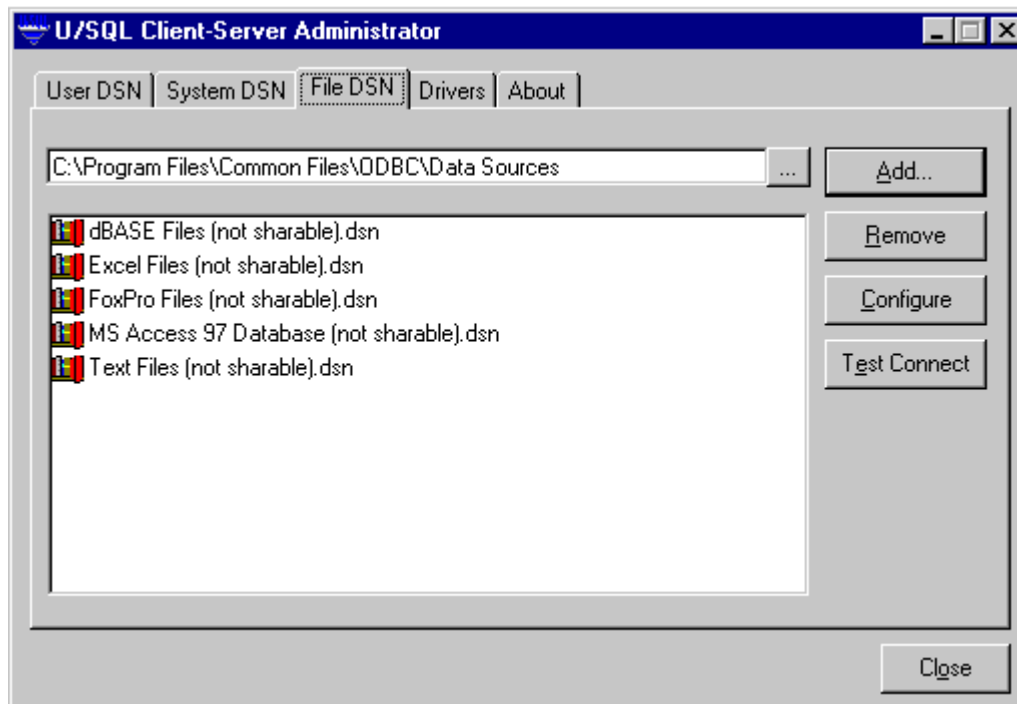


Enter the name of the File DSN you want to create (you do not need to include the '.dsn' extension as this will be added automatically), and select the **Create new Data Dictionary** option. Click **Next** and follow the instructions to create the File DSN. It will be created in the default Microsoft directory for File DSNs.

To create a new File DSN using an existing data dictionary, enter the name of the File DSN you want to create, and select the **Attach an existing Data Dictionary** option. Click **Next**. Enter the details of your existing data dictionary and follow the instructions to create the File DSN. It will be created in the default Microsoft directory for File DSNs.

U/SQL Administrator

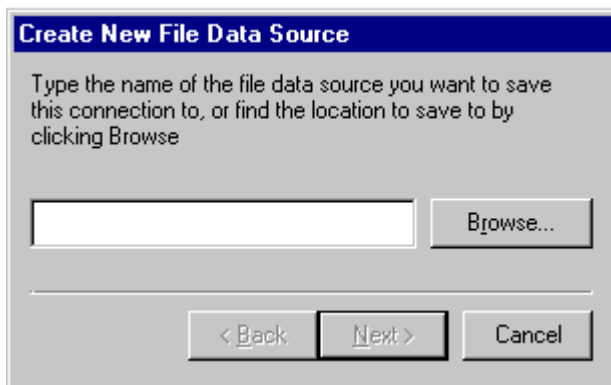
Run the U/SQL Administrator and click the **File DSN** tab. The **File DSN** property page is displayed:



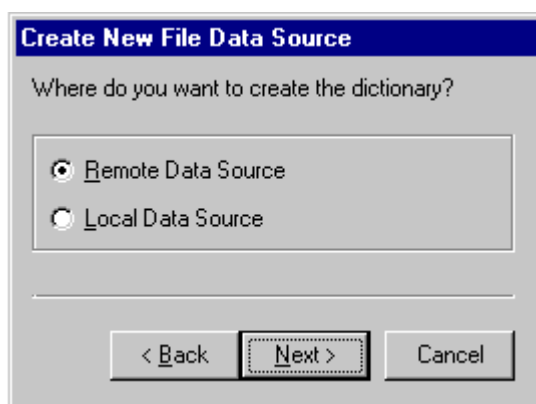
This property page allows you to:

Add a new File DSN

Click **Add**. The **Create New File Data Source** dialog box is displayed:



Enter the name of the File DSN you want to create (you do not need to include the '.dsn' extension as this will be added automatically), and click **Next**. The following dialog box is displayed:



Select the **Remote Data Source** option if you are using the Multiple-tier version of U/SQL .

Select the **Local Data Source** option if you are using the Single-tier version of U/SQL.

Follow the on-screen instructions to create the File DSN.

Remove a File DSN

Select the File DSN you want to delete and click **Remove**. You will be asked to confirm that you want to delete the specified file. Click **Yes** to delete the file.

Note: *This does not delete the data dictionary.*

Configure a File DSN

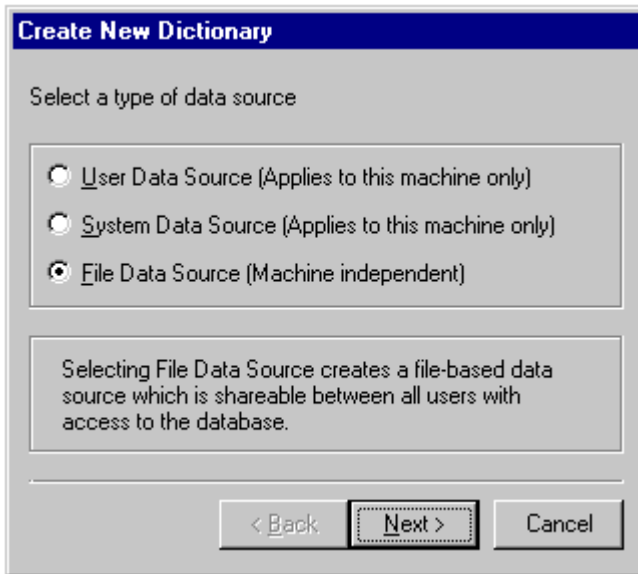
Select the File DSN you want to configure and click **Configure**. The **32-bit ODBC Setup** dialog box is displayed allowing you to configure the File DSN.

Test a connection to the File DSN

Select the File DSN you want to test and click **Test Connect**.

Win U/SQLi

Run the [Win U/SQLi](#) utility and select the **New UDD** command from the **File** menu. The **Create New Dictionary** dialog box is displayed:



This dialog box contains the following options:

| | |
|---------------------------|---|
| User Data Source | The User Data Source option is selected by default. This creates a data source, which is specific to your machine, and visible only to you. |
| System Data Source | Selecting the System Data Source option creates a data source, which is specific to your machine, and usable by any user who logs onto your machine. |
| File Data Source | Selecting the File Data Source option creates a data source, which is independent of your machine. |

To create a File DSN select the **File Data Source** option. The **Create New File Data Source** dialog box is displayed:



To create a new File DSN, enter the name of the File DSN you want to create (you do not need to include the '.dsn' extension as this will be added automatically), and select the **Create new Data Dictionary** option. Click **Next** and follow the instructions to create the File DSN. It will be created in the default Microsoft directory for File DSNs.

To create a new File DSN using an existing data dictionary, enter the name of the File DSN you want to create, and select the **Attach an existing Data Dictionary** option. Click **Next**. Enter the details of your existing data dictionary and follow the instructions to create the File DSN. It will be created in the default Microsoft directory for File DSNs.

Single-tier Administration

Installed Files

When you install Single-tier U/SQL the installation setup program performs the following tasks:

- It checks that you have the Microsoft ODBC Driver Manager installed (as a minimum version 2.5, **ODBC32.DLL**) in the **\WINDOWS\SYSTEM** directory. If you do not have this, the setup program will install it.
- It installs the U/SQL ODBC Driver (**TSENG32.DLL**), and the other U/SQL Server DLL's (for ACUCOBOL, **TSACU32.DLL**; for Micro Focus COBOL, **TSMF32.DLL**) in the **\WINDOWS\SYSTEM** directory.
- It installs the [U/SQL Administrator](#), which is used to set up the **ODBC.INI** directive entries which are described later in this section.
- If ordered, it installs the U/SQL Manager which is used to create, view, amend or delete the UDD for COBOL data sources. It can also be used to set up the **ODBC.INI** directive entries.
- Optionally, it installs a demonstration application, written in Microsoft Visual Basic, for a Books Wholesaler. This demonstration shows examples of various queries, graphics and OLE to Microsoft Excel spreadsheets. The Visual Basic source is provided to help you review the application. The sample data files and corresponding UDD are included so that the application can be run.

For the Books demonstration, it edits the **ODBC.INI** Registry entries to add a section for the **BOOKSW32.UDD** data source and dictionary.

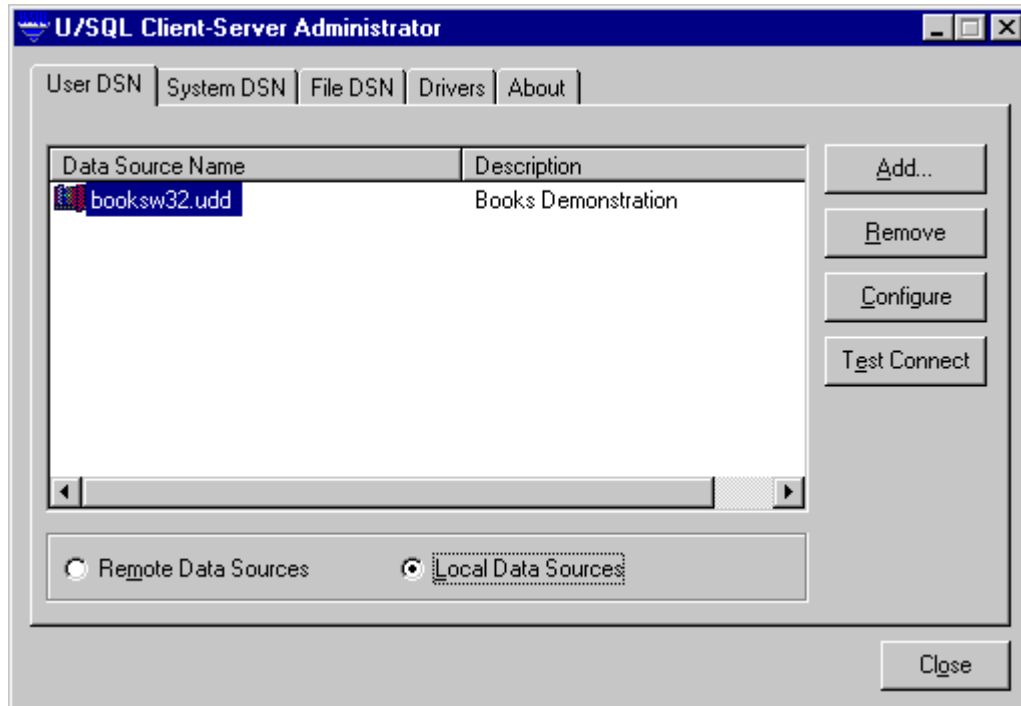
- Optionally, it installs the interactive U/SQL utility, [Win USQLi](#), that allows sample queries to be made, provides export/import functions for UDDs and allows you to query the message log file. This is a simple ODBC-enabled product that you can use to query the Books Demonstration data or your own application tables.
- It installs the [License Tool](#) that is used to install and examine the License.
- It creates a set of U/SQL Client icons in the appropriate Windows program group.

At this point, the installation of the Single-tier U/SQL software is complete.

U/SQL Administrator

For non-COBOL Single-tier versions, or where you have already created a UDD, further ODBC entries can be set-up using the **U/SQL Administrator**.

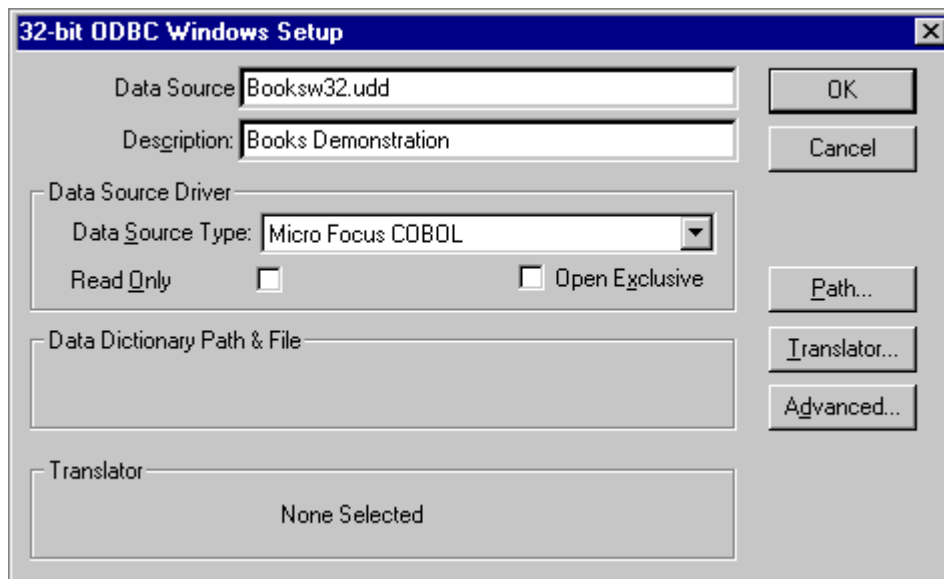
Invoke the **U/SQL Administrator**. The **U/SQL Administrator** property sheet is displayed:



Ensure, for Single-tier, that you select **Local Data Sources**. These options are only displayed if you have both Single and Multiple-tier U/SQL ODBC Drivers on your PC.

Entering the ODBC.INI Directives

From the [U/SQL Administrator](#) property sheet, click **Add**. The **32-bit ODBC Windows Setup** dialog box is displayed:



You then make the entries shown, as described in the section [ODBC.INI Directives](#):

- **Data Source Name:** This is only entered here when using the Administrator, for example booksw32.udd. The '.udd' extension is NOT mandatory for the Single-tier version.
- **Description of the Data Source.**
- **Data Source Type:** Usually there will be only one, otherwise make your selection.
- **Read Only:** Select this check box if this option is required.
- **Open Exclusive** (for data files): Select this check box if this option is required. This is only applicable to Micro Focus COBOL to improve performance.

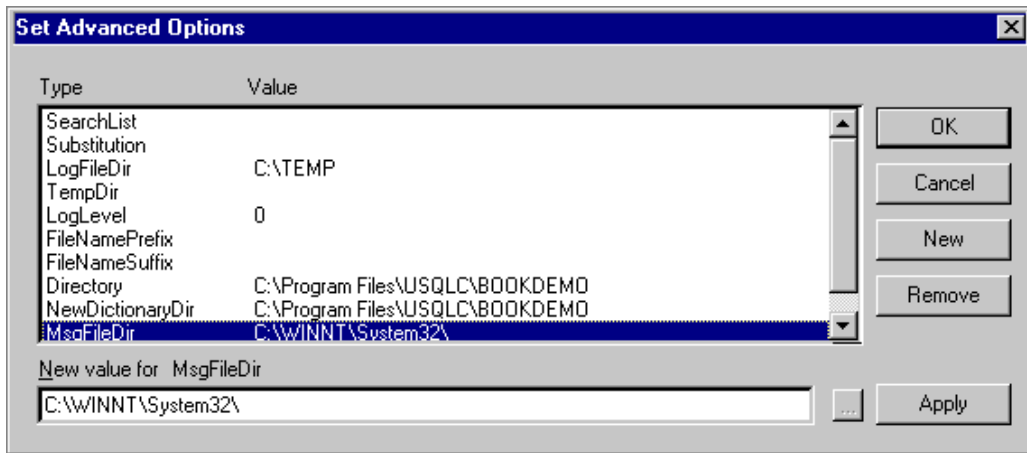
Note: You must ensure that when creating or modifying a UDD using the U/SQL Manager that **OpenExclusive** is not set.

- **Data Dictionary Path and File name:** The path defaults to the directory where U/SQL Adapters was installed and the dictionary name to the Data Source Name, with a '.udd' extension automatically added. You can change this by clicking the **Path...** button to display a browser to change the path and UDD name.

Note: As this is a dictionary name, not a data source name, the '.udd' extension is mandatory.

These are the minimum **ODBC.INI** entries that are automatically added to the Registry when you click **OK**. Additional entries can be made as follows:

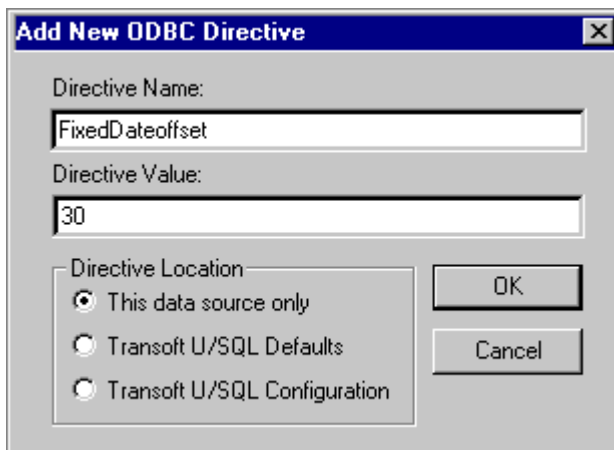
- **Advanced** entries. Click **Advanced...** . The **Set Advanced Options** dialog box is displayed:



You can select any of the options under **Type** which then displays any existing **Value** in the lower edit box allowing you to amend it or make a new entry. After changing an entry, click **Apply** and a (*) appears alongside the Type description, indicating a change has been made. When all changes have been applied, click **OK**.

To remove a certain directives, highlight it and click **Remove**.

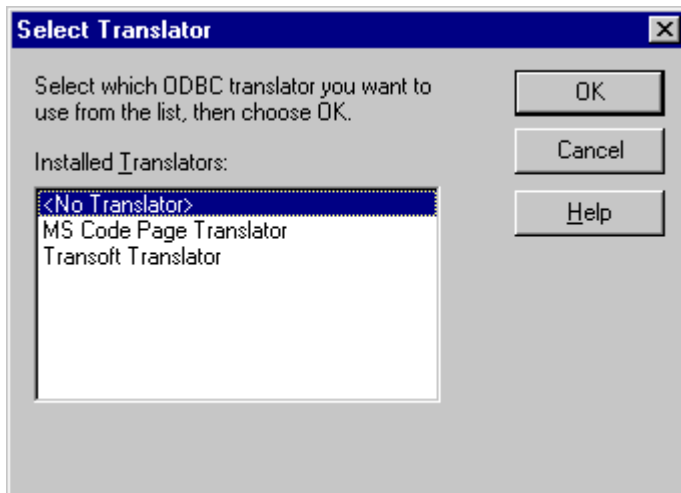
To include directives not shown under **Type** click **New**. The **Add New ODBC Directives** dialog box is displayed:



Enter the new **Directive Name** and **Directive Value** and click **OK**. For details on valid entries refer to the [ODBC.INI Directives](#) section. You also can define the directive location, either in the **[Transoft U/SQL Configuration]** section or in the particular data source **[<Data_source_name>]** section.

- **Translator** entries. ODBC translators can be added to modify the data from the Driver, that is the client, to the Data Source and from the Data Source to the Driver.

Click the **Translator...** button in the **ODBC Windows Setup** dialog box. The **Select Translator** dialog box is displayed:

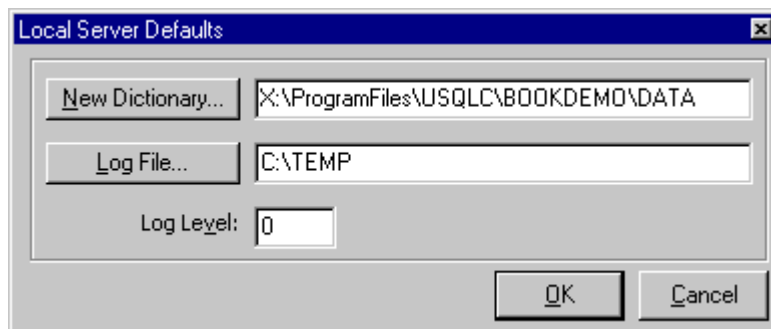


To select an ODBC translator from those installed on your PC, either double-click the required translator or highlight the one required and click **OK**.

If you click on the Transoft Translator, then after returning to the ODBC Setup dialog box you will find a browse button in the Translator section, which you can use to select the translation table you want to use. See the section [Foreign Character Set Support](#).

Local Server Defaults

In the U/SQL Manager for COBOL data sources, select the **Local Server Defaults** command from the **Connection Options** menu. The **Local Server Defaults** dialog box is displayed:



These entries create the [\[Transoft U/SQL Configuration\] section](#) in the **ODBC.INI** Registry. Click the **New Dictionary** or **Log File** buttons to assist in establishing the appropriate directories.

Multiple-tier Administration (Windows)

U/SQL Server Installation on Windows

U/SQL Directories and Files

The following directories and files are relative to the base directory of your U/SQL Server software installation, which by default is **C:\USQLCS**:

| Directory (relative to U/SQL base) | File | Function |
|--|---|--|
| <base> C:\USQLCS (By default) | RELEASE.WRI | Release notice. |
| \BIN | USQLSD32.EXE | U/SQL Server. |
| | TSMENG32.DLL | U/SQL Engine DLL. |
| | TSM*32.DLL | Data Source Driver DLL. For example, for Micro Focus COBOL it is TSMMF32.DLL . |
| | USQLSM.EXE | U/SQL Service Manager. |
| | WIN32RAP.DLL | U/SQL Service Manager ancillary functions DLL. |
| | CLIPPING.EXE | A diagnostic program which allows your TCP/IP connection to be tested. This is only used if your PC application appears to have failed to connect. It is used with the TestNet utility on your Windows PC. See the Troubleshooting section. |
| | USQLCS.MSG | The U/SQL Server error messages file. |
| | TSTRANNT.DLL | DLL providing language translation functions. |
| | *.TRN | Foreign character set conversion files. See the section Foreign Character Set Support . |
| | TSTRANS.DAT | Skeleton translation table file. |
| \BOOKDEMO | CUSTOMER etc | Books Wholesaler data files. |
| | BOOKS.UFD | UFD import file to create the BOOKS.UDD . |
| \UDD | Directory where UDDs will be created, by default. | |

| | | |
|--|-----------|---|
| | BOOKS.UDD | The dictionary (UDD) for the Books Wholesaler demonstration . |
|--|-----------|---|

U/SQL Service Manager

For Windows NT Server and Windows 2000, the 32-bit **ODBC.INI** entries are stored in the Windows Registry. The **U/SQL Service Manager** utility is used to set up and amend the Section Names and directives.

Note: *It is recommended that you don not use the Registry editor, **regedt32**, to add or modify entries as it is possible to make mistakes; use the U/SQL Service Manager instead.*

The **U/SQL Service Manager** utility allows you to:

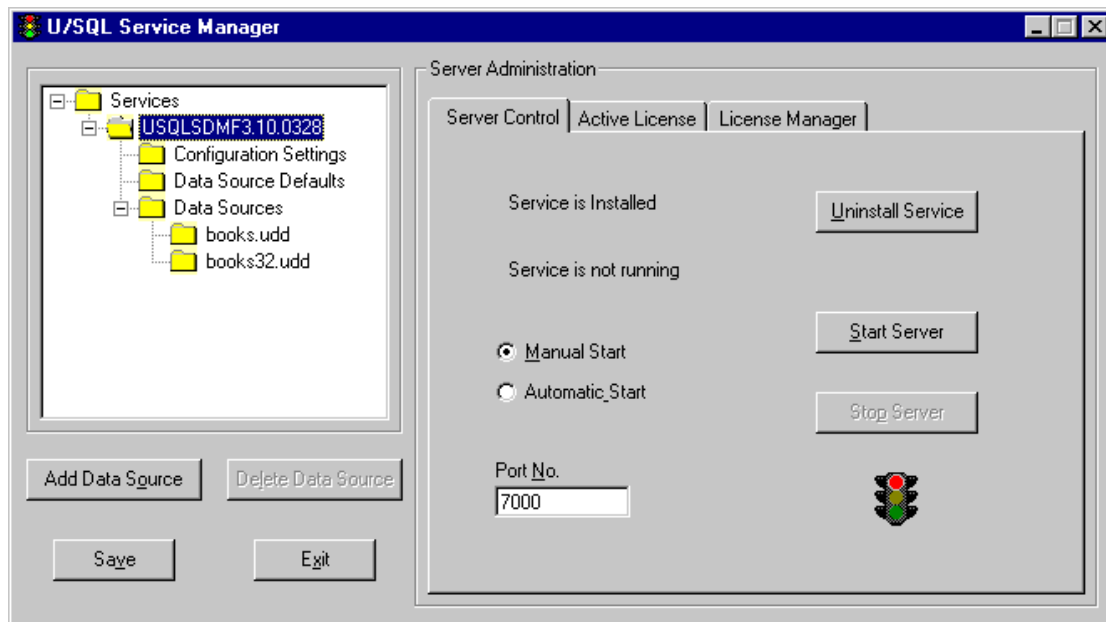
- Start the U/SQL Server as a Windows NT Server or Windows 2000 Service and make this automatic on start up of the Windows NT or Windows 2000 machine or have to be actioned manually, when required.
- Stop the Windows U/SQL Server.
- Install and activate your U/SQL Server license. Refer to the [Installing the Server License on Windows NT Server](#) section.
- Set up the Configuration Settings, the Data Source Defaults and the individual [Data Source directive settings](#).

When you install the U/SQL Server software, a Service folder is created in the Registry, with sub-folders for Configuration Settings, Data Source Defaults and an individual Data Source folder for **books.udd**, the Books demonstration application. These folders are populated with various directive values as a result of the installation process.

You can have multiple revisions of U/SQL Server software resident on the same machine with each given a separate Service name, of the form **USQLSDXXXN.NN.NNNN**. The **XXX** indicates the data source driver installed, for example, ACU for ACUCOBOL, MF for Micro Focus COBOL or BB for BBasic ISAM; the **N.NN.NNNN** indicates the revision of U/SQL Server installed.

The following property pages show the functions provided by the U/SQL Service Manager. Refer to the [U/SQL Server Directives](#) section, for the various directive settings.

Server Administration



The **Server Control** property page is displayed by clicking on the name of a Service, for example, USQLSDMF3.10.0328.

This property page contains the following fields:

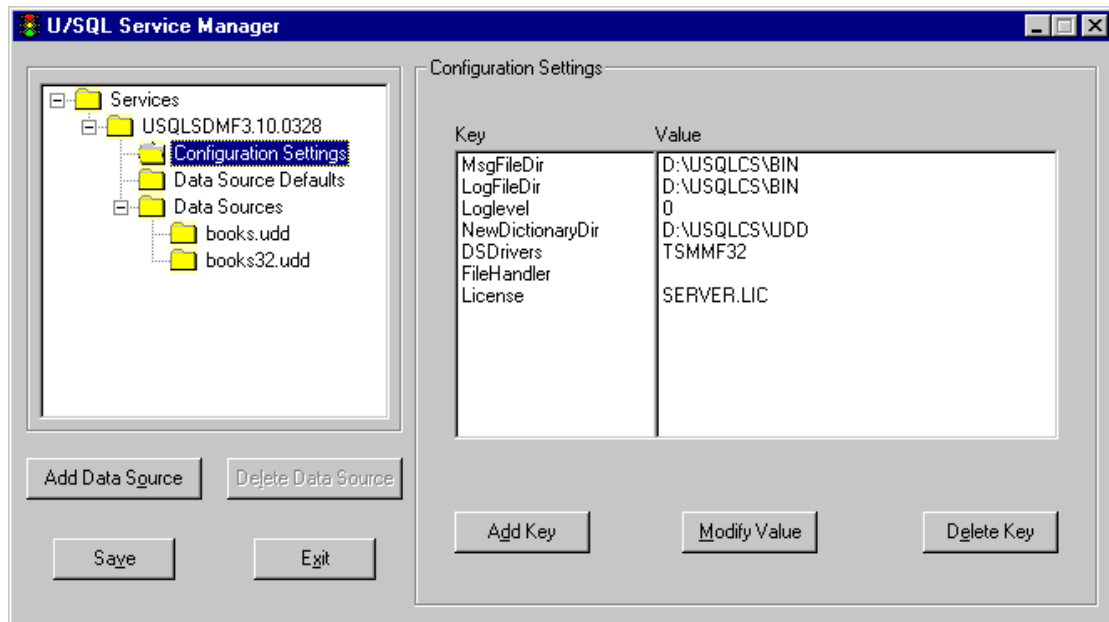
| | |
|--|--|
| Port No. | The port number is allocated during installation. The Port number cannot be changed while the service is running. |
| Manual Start Automatic Start | If you select Automatic , the U/SQL Server starts automatically as a service process when the Windows NT Server or Windows 2000 machine itself is started If you select Manual , you have to start the U/SQL Server via this utility using the Start Server and Stop Server buttons. |
| Install Service Uninstall Service | To uninstall U/SQL Server software: <ul style="list-style-type: none"> • Stop the service. • Click the Uninstall Service button to remove the registry entries for the service. <p>Note: Once uninstalled this button is renamed Install Service and you click it to re-install the service.</p> <ul style="list-style-type: none"> • Then use the uninstallShield utility in the program group. |
| Start Server Stop Server (Pause Server) | If you have selected Manual startup, then to start the U/SQL Server service click Start Server . The traffic light symbol changes from red to green to indicate that the service has started: |



All Server Control changes take effect immediately, except Port number which cannot be changed while the service is running. All other changes to the directives only take effect when the **Save** button is clicked.

Configuration Settings

Clicking on the **Configuration Settings** tab, for a particular service, displays the **Configuration Settings property** page:



This contains the directives: **MsgFileDir**, **LogFileDir**, **LogLevel**, **NewDictionaryDir** and **License**. Their default values (except License) are set up during the U/SQL Server software installation but you can change them to meet your requirements. The **DSDrivers** (Data Source Drivers) is defined in this form, but this is also automatically set up during installation and must not normally be changed.

To add a new directive click **Add Key**.

To change a directive's value, select the directive and click **Modify Value**.

Normally, you do not need to remove a directive and its value, but to delete a directive, select it and click **Delete Key**.

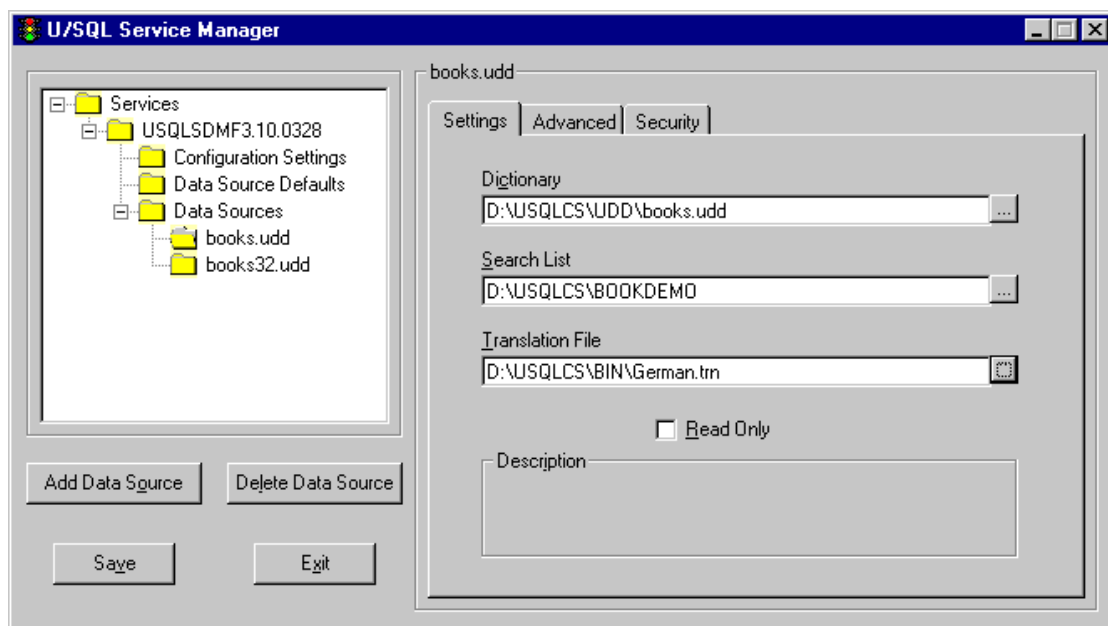
After highlighting a Service, a new Data Source can be added by clicking the **Add Data Source** button and entering the new name in the edit box provided

terminated by the RETURN key. A highlighted data source is removed by clicking the **Delete Data Source** button.

Click on an existing data source to view or modify its directive values. For example, select the **books.udd** data source. These are presented on three property pages.

Note: *The same directives, under the three tab forms, can be set up as Data Source Defaults to apply to all data sources unless overridden by entries in individual Data Sources.*

Settings

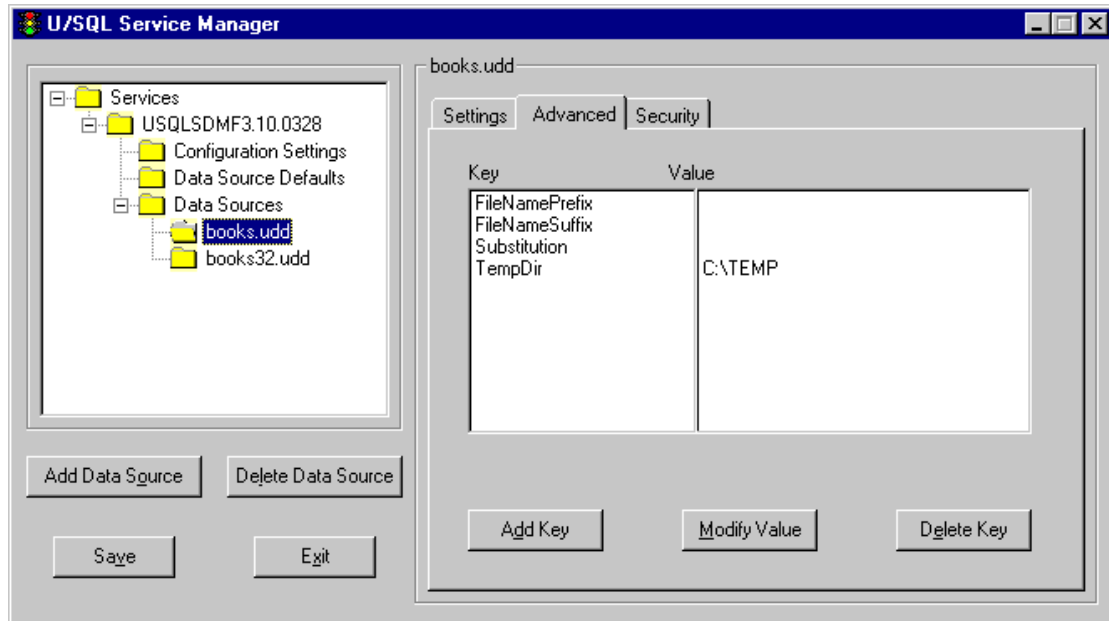


The Settings property page for a Data Source is used to set up the key directives; the Dictionary, the Search List, any Translation File and whether you wish to allow Read Only access to the data files. A Description can be entered in the box at the bottom of the form.

The **TranslationDLL** directive of **TSTRANNT.DLL** is automatically set up.

Advanced

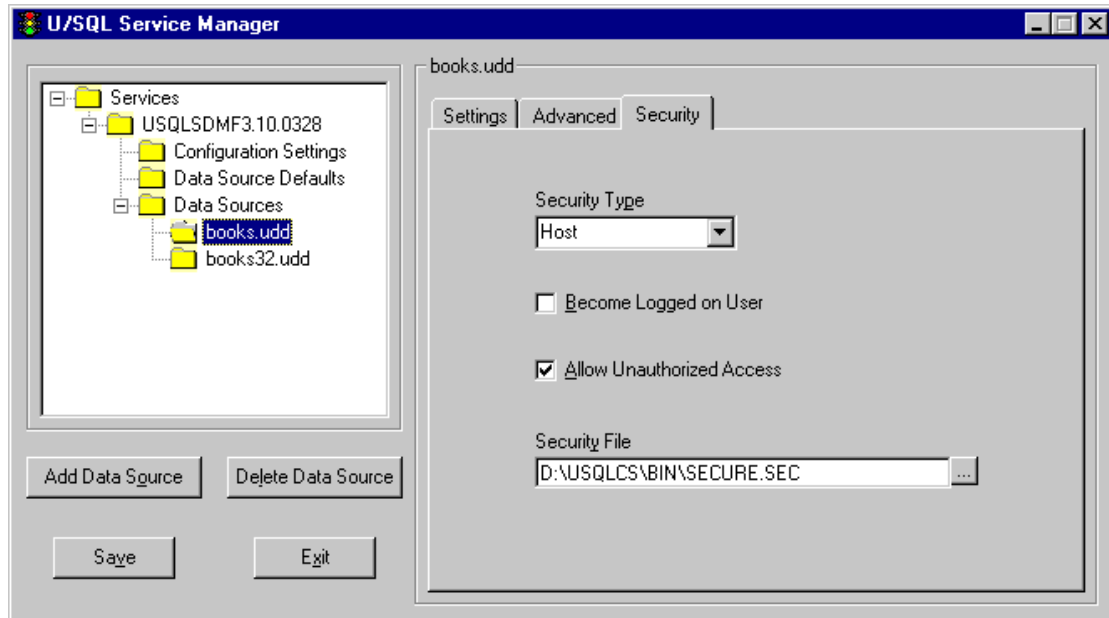
Click the **Advanced** tab for the Data Source to display the **Advanced** property page:



This allows you to optionally enter: Directory where the U/SQL Server will start, **Substitution**, **FileNamePrefix**, **FileNameSuffix** and **TempDir**. To add directives click **Add Key**. To delete a highlighted entry click **Delete Key**. To change a highlighted directive's value click **Modify Value**.

Security

Click the **Security** tab to display the **Security** property page:



This property page allows you to modify the Security settings. See the [Security](#) section.

U/SQL Server Directives

The U/SQL Server requires configuration directives on the Windows NT Server, or Windows 2000 platforms. There must be entries for each UDD set up on the server. These entries include:

- The Data Source Name (DSN), by which the UDD and its associated directive entries are known. The DSN is usually the name of the UDD.
- The UDD name with its path, if the DSN is not the UDD name.
- Optionally, there can be entries for read only, a searchlist for the data files, file name substitutions, user connection level security and logging information.
- Other optional entries.

Note: *It is important that the Data Source Names used in the U/SQL Server configuration directives match the **ODBC.INI** settings for the U/SQL Client, see the section [Client ODBC.INI Directives](#), otherwise the client and server will not be able to connect.*

Where U/SQL Server Directives Reside

For Windows NT Server and Windows 2000, the **ODBC.INI** entries are contained in Registry folders. The [U/SQL Service Manager](#) utility, in the Windows U/SQL Adapters program group, is used to set up the Section Names, as folders, and directives.

Note: *Do not use the Registry editor, **regedt32**, to add or change directives, as it is very easy to make a mistake. Always use the U/SQL Service Manager.*

The U/SQL Server Directives

The U/SQL Server directives are grouped under three Section Names:

- Configuration Section
This Section Name defines the global settings for the server for all data sources.
- Data Source Defaults
This Section Name defines the default directives for all data sources unless overridden in individual data source section(s), below.
- <data_source>
This Section Name is the name of a specific data source, for example, <data_source> could be 'books.udd'. There can be any number of data source entries (that is, UDDs) in the settings, describing different applications you may wish to connect to, or different views of the same application.

The following sections detail the directives that can be set under each of the three Section Names in either the **usqlsd.ini** configuration file for UNIX platforms or in the **USQLSD.INI** Registry folders for Windows NT Server and Windows 2000.

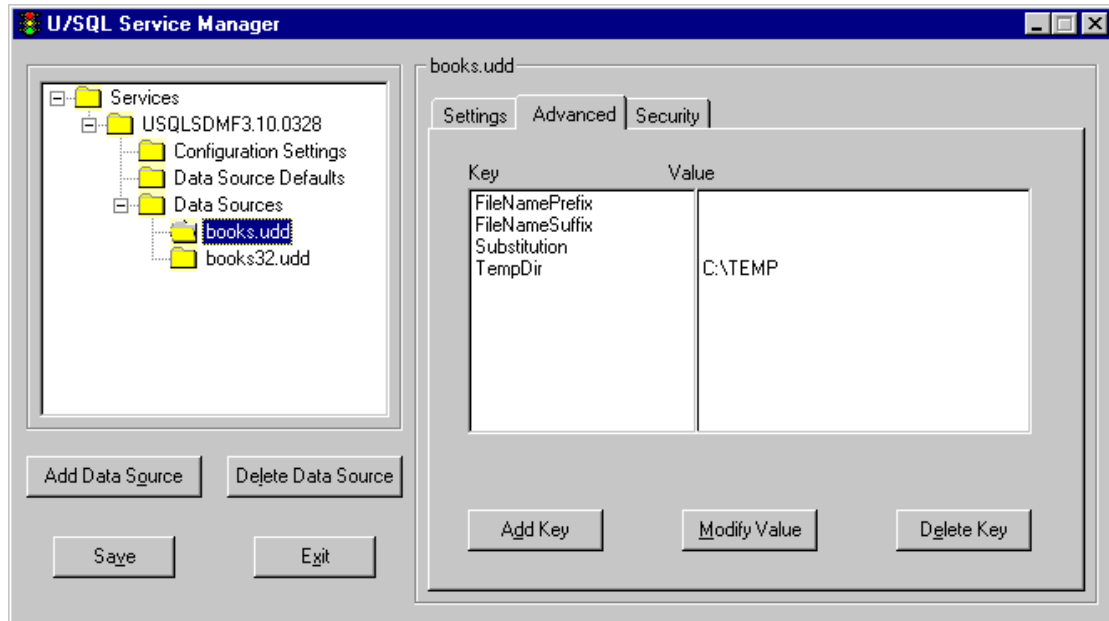
- Configuration Section

- Data Source Defaults Section
- [<data source> Section](#).

Note: *It is very important that the exact spelling, and this includes use of mixed lower and UPPERCASE where appropriate, of both the directives and their values is adhered to.*

Example U/SQL Directives

The following is an example of directives viewed via the [U/SQL Service Manager](#) under Windows NT Server.

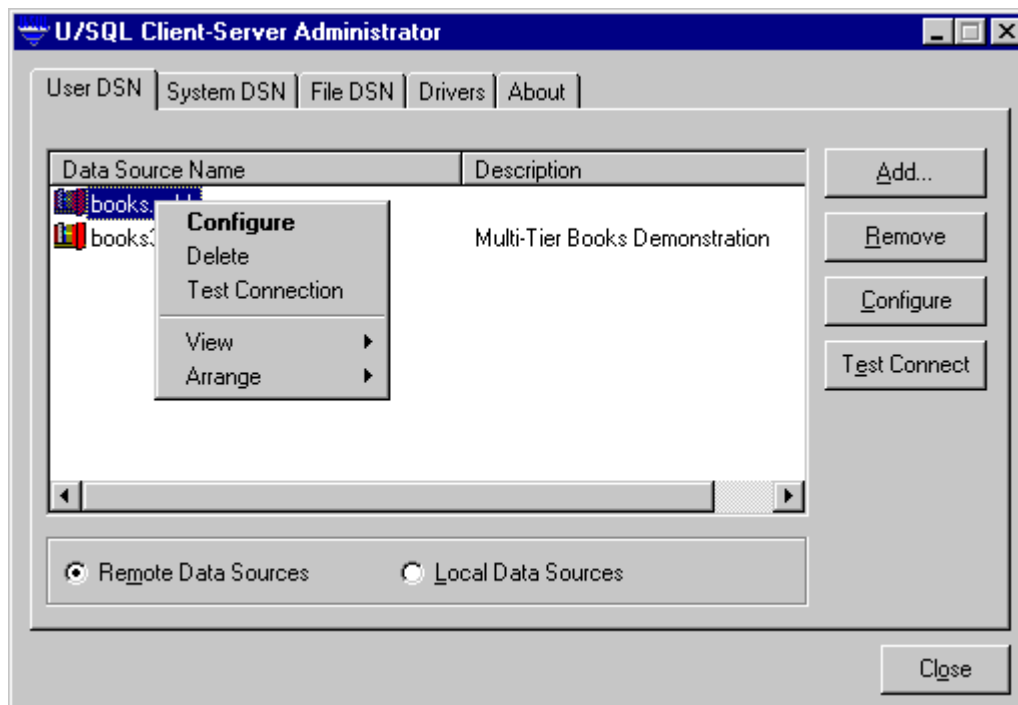


U/SQL Administrator Facilities

The **U/SQL Administrator** has a number of facilities.

- You can [test connect to a UDD](#) to ensure you have setup the directives correctly, before attempting to use an ODBC-enabled product. If you fail to connect details of the connection failure are displayed.
- You can [view the ODBC drivers installed](#) to ensure you have the correct one(s) installed.
- You can invoke the [System & File Information utility](#). This provides useful information to check that your U/SQL Client is installed successfully.

You can use the right mouse button to obtain different views of the Data Sources, for example, as larger icons. Clicking the right mouse button after highlighting a data source, provides an alternative menu to the **Remove**, **Configure** and **Test Connect** buttons:



Note: If Microsoft's ODBC Driver Manager 3.0 or later is detected on your PC, then a **System DSN** tab in addition to a **User DSN** tab are included in the U/SQL Administrator property sheet, otherwise only a single **Data Sources** tab is provided. Refer to the section [Adding ODBC.INI Entries](#).

TestNet

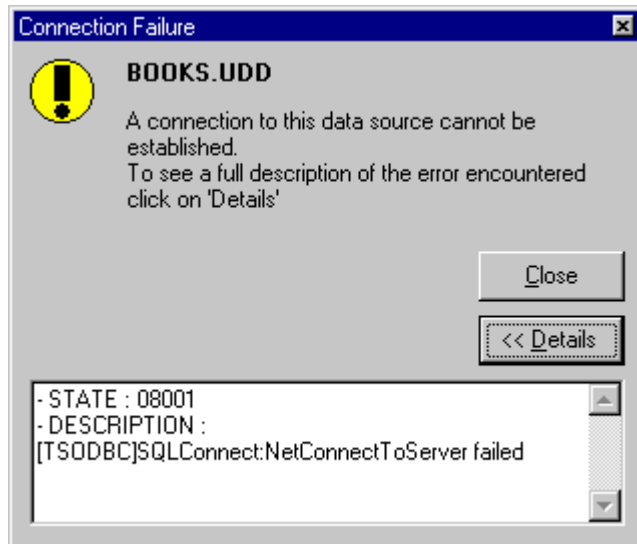
You can test you local PC's configuration by clicking the **TestNet** button on the U/SQL Administrator property sheet.

To test the network you will need to have the **clipping** utility invoked on the server.

Test Connect to a UDD

When you have added a new UDD data source it is useful to ensure that the setup is correct. To do this click the **Test Connect** button on the U/SQL Administrator property sheet. If successful, then the a message informing you that the connection has succeeded is displayed.

If the connection fails then the following message box is displayed:

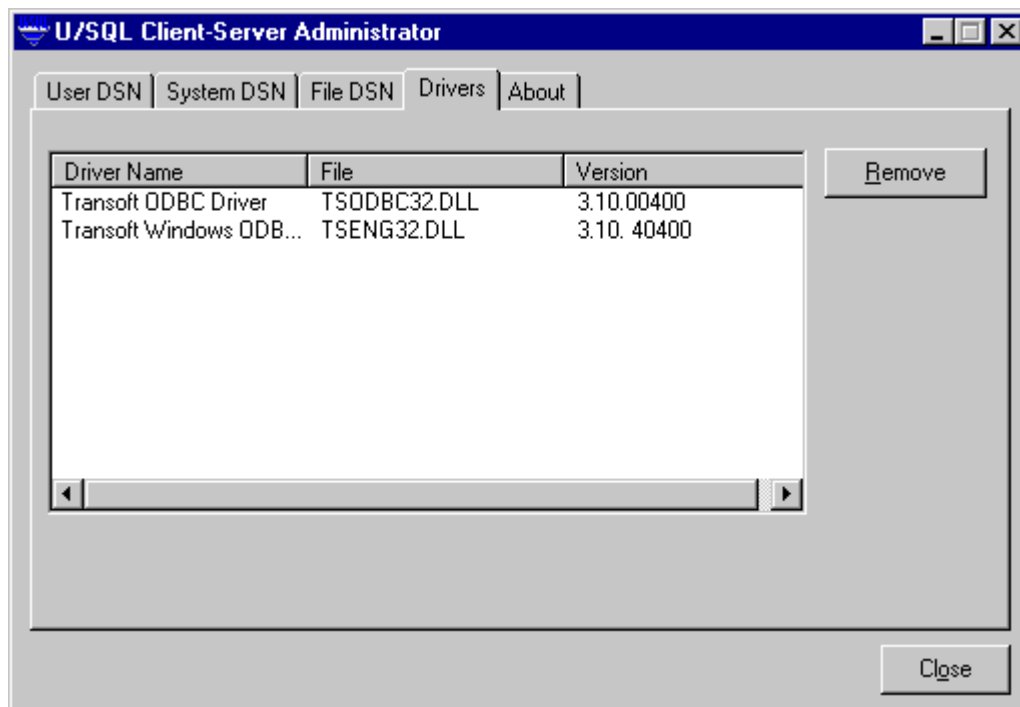


Click **Details** for a full description of the error. This details of the error are displayed in a text box as can be seen above.

If there is an error, the two most common reasons are either that the server is not started, or the details entered for server and/or port are incorrect. Check these settings and then see the Troubleshooting Guide for more information if there is still a problem.

View ODBC Drivers installed

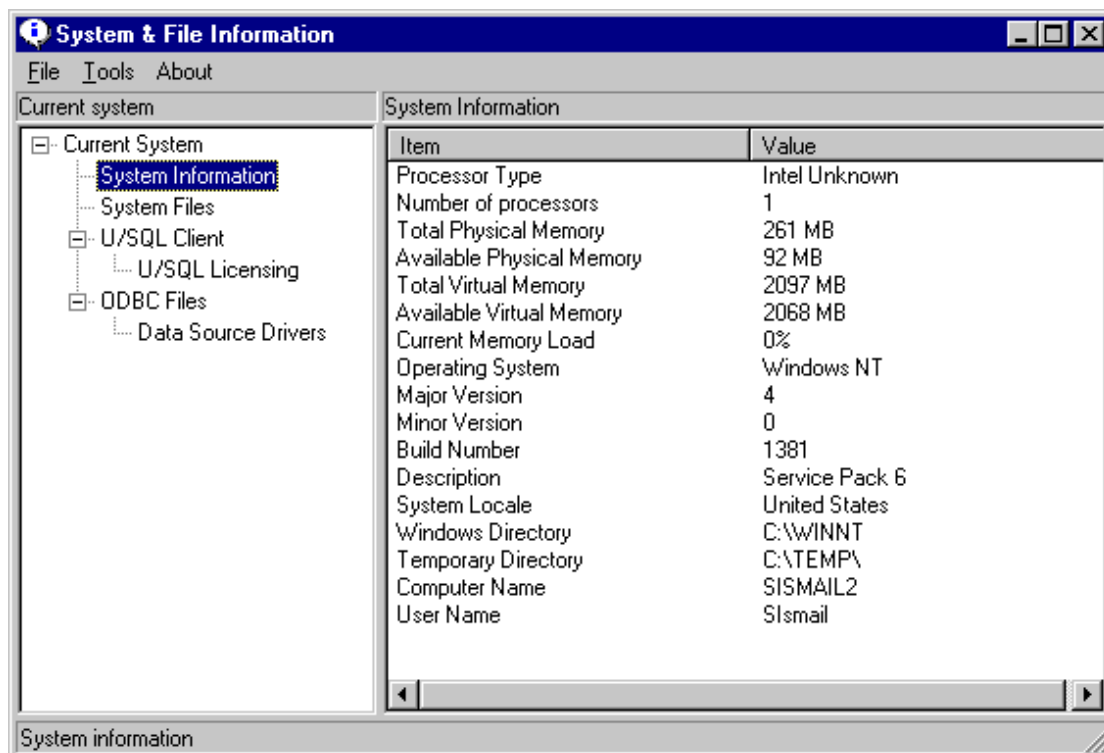
Select the **Drivers** tab, on the U/SQL Administrator property sheet to view the Transoft U/SQL ODBC Drivers installed and their Version numbers:



The **Transoft Windows ODBC Driver** is the Single-tier ODBC driver, while the **Transoft ODBC Driver** is the Multiple-tier ODBC driver. To delete either driver, select the driver and then click **Remove**.

System & File Information Utility

Select the **About** tab, on the U/SQL Administrator property sheet, and click **System Info**. The **System & File Information** dialog box is displayed:



Alternatively select the **File & System Information** icon from the **Start** menu.

Transoft U/SQL User Guide

This utility provides useful checking information, for example, to ensure that the Micro Focus COBOL **TSMF32.DLL** Data Source Driver can be loaded, simply click on it and if successful then the Can Load column will change from Unknown to Yes.

Multiple-tier Administration (UNIX)

U/SQL Server Installation on UNIX

Once you have completed the U/SQL Server software installation on UNIX, the following files and directories are installed on your system.

U/SQL directories and files

The following directories and files are relative to the base directory of your U/SQL Server software installation, by default **/usr/usqls**:

| Directory (relative to U/SQL base) | File | Function |
|--|-------------------------------------|---|
| <base> | README | Release notice. Read using: <code>more README</code> |
| | install.data | Details of the installation including the TCP/IP socket port to be used. |
| | install.sh | The installation script you have just used. |
| | .chksum | The checksums for all the installation files. They are used to ensure that each file has not been corrupted in any way. |
| | .disc_dev | The media device used for the installation. |
| /bin | The U/SQL Server software directory | |
| | Executables | See the section U/SQL Server Executables . |
| | Scripts | See the section U/SQL Server Scripts . |
| | usqlsd.ini | The U/SQL Server configuration file. |
| | usqlcs.msg | The U/SQL Server error messages file. |
| | *.dat | Directives control files. |
| | tstrans.dat | Skeleton translation table file. See the section Foreign Character Set Support. |
| | *.trn | Foreign character set conversion files. |
| /lib | *.a *.o | For certain Data Sources only. Library modules of the U/SQL Server. |
| | makefile | For linking the U/SQL Server to the data source file handler. Refer to the Installing the Multiple-tier U/SQL Server section. |

| | | |
|-----------------|--|--|
| /example | This directory contains the data files for the Books Wholesaler demonstration. | |
| | books.udd | The dictionary (UDD) for the Books Wholesaler demonstration. |
| | books.ufd | The UFD import file to create the books.udd . |

U/SQL Server Executables

The following executables are included with the U/SQL Server software and are found in the **bin** directory, by default **/usr/usqls/bin**.

| Executable | Function |
|------------|--|
| usqlsd | The U/SQL Server daemon. |
| usqli | The Interactive U/SQL server based client utility. This utility allows SQL syntax to be executed interactively. See the usqli section. |
| cliping | A diagnostic program which allows your TCP/IP connection to be tested. This is only used if your PC application appears to have failed to connect. It is used with the TestNet utility. |
| freepport | Finds the next free socket port number. |

U/SQL Server Scripts

The following scripts are included in the U/SQL Server installation, or created when **install.sh** is run, and are found in the **bin** directory, by default **/usr/usqls/bin**:

| Script name | Function |
|---------------|---|
| serv_setup.sh | This is called by the install.sh script but can be run again at any time to: <ul style="list-style-type: none"> Maintain the usqlsd.ini configuration file. It uses the ConfigSet.dat and DatSrcDefs.dat directive prompts files. View the install.data file showing the socket port number. |
| start_serv.sh | Starts the U/SQL Server defaulting to the port number that was determined at install time. Alternatively the next freepport or any other port number can be used. |
| stop_serv.sh | Stops the U/SQL Server running. It displays all the Servers running and requests the port number of the Server you wish to stop. |

| | |
|--|---|
| check_serv.sh | Checks if the Server is running. If it is, then the port number and PID are reported. |
| cobol_export.sh cobol_import.sh cobol_rebuild.sh | These scripts are used to export and import COBOL UDDs. See the section Modifying a UDD . |

You normally invoke these scripts from the **bin** directory, for example:

```
cd /usr/usqls/bin
./serv_setup.sh
```

You can also display the contents of these scripts by performing the following:

```
more <script_name>
```

or

```
pg <script_name>
```

The U/SQL Server

The U/SQL Server program, **usqlsd**, is a fully daemonized server. This means that it will automatically become a background process, disassociating itself from the current process group and user terminal. No special start-up script is needed to do this and it can be directly run from a terminal or, more typically, from a simple **rc** start-up script.

The purpose of the initial Server daemon is to wait for a client connection request on a pre-specified TCP/IP socket port. After a successful TCP connection is established, the Server daemon forks a child process to exclusively handle this connection for the client. When the connection is closed, the server process for that client terminates. There is therefore one new server process for each client ODBC connection made.

Starting and Stopping the UNIX Server

Starting the UNIX U/SQL Server

To start the U/SQL Server you will need to have a valid server license installed and defined by the License directive in the [usqlsd.ini configuration](#) file. Refer to the [Installing the Server License on UNIX Platforms](#) and [Configuring Multiple-tier U/SQL](#) sections.

Typically the U/SQL **usqlsd** Server daemon is automatically initiated by an **rc** start-up script, at system boot time. This is defined during the initial U/SQL Server software installation.

It is possible, however, to start the U/SQL Server process at any time from a user terminal. To do this, execute the **start_serv.sh** script, from the **bin** directory below the base directory of the U/SQL Server software installation, for instance, **/usr/usqls/bin** as follows:

```
cd /usr/usqls/bin
./start_serv.sh
```

This can be useful for initial testing and for special circumstances such as manually restarting a terminated U/SQL Server.

Note: *Ensure the U/SQL Server is started with a username and group ID which have the permissions required to access the UDD and data files. It is recommended that the U/SQL Server process be started as UNIX root, which will then enable it to change to the user ID specified in the ODBC connection. This results in a more secure system. See the section [Multiple-tier Security](#).*

Stopping the UNIX U/SQL Server

You can stop the U/SQL Server at any time by running the **stop_serv.sh** script, again from the **bin** directory:

```
cd /usr/usqls/bin
./stop_serv.sh
```

This displays details of the U/SQL Servers running and request the port number of the Server(s) you would like to shut down. Once a port number is entered, all U/SQL Server(s) with the same port number stop with the message:

The server has been shutdown successfully.

Setting up the UNIX `usqlsd.ini` File

The UNIX `usqlsd.ini` configuration file can either be managed manually using a suitable editor (for example, `vi`) or normally by the `serv_setup.sh` script, which is located in the `bin` directory under the U/SQL Server software installation directory, which is by default `/usr/usqls/bin`. The `usqlsd.ini` file is maintained in the `bin` directory.

You must be signed on as **superuser** to invoke the script in the `bin` directory:

```
cd /usr/usqls/bin
./serv_setup.sh
```

The **Main Menu** is displayed:

```
U/SQL SERVER CONFIGURATION AND CONTROL - MAIN MENU
=====
```

- 1) DATA SOURCE CONFIGURATION
- 2) CONFIGURATION SETTINGS
- 3) START SERVER
- 4) STOP SERVER
- 5) LOG FILE CONTROL
- 6) CREATE, VIEW & REMOVE SERVER LICENSE
- 7) VIEW THE 'usqlsd.ini' FILE

Enter '?' or '<NUMBER>?' for HELP, 'x' to EXIT.

Select option:

To add or modify the entries for any Data Source, select **1** from the Main Menu. The **Data Source Configuration** menu is displayed:

```
DATA SOURCE CONFIGURATION - MENU
```

- 1) Modify Data Source Defaults
- 2) Add a new Data Source
- 3) Modify a Data Source
- 4) Delete a Data Source
- 5) List Data Sources
- 6) View the 'usqlsd.ini' file

Enter '?' or '<NUMBER>?' for HELP, 'x' to Return to Main Menu.

Select option:

- Select **1** to modify the [**Data Source Defaults**] section.
- Select **2** to add a new [**<Data_source_name>**] section. You are first requested to enter the new UDD name. This does not have to have a '.udd' extension. For example, you may have two data sources, Company1

and Company2, which both use the same UDD, company.udd, see the [Multi-company Support](#) section.

You are then prompted for all the other possible data source directives.

- Select **3** to modify the existing entries of a data source. You cannot add new entries. To do this, first delete the data source (Select **4**) and then add a new data source (select **2**).
- Select **4** to delete a data source; select **5** to display a list of all the data sources; and select **6** to view the contents of the **usqlsd.ini** configuration file.

After returning to the Main Menu, to modify the entries in the **[Configuration Settings]** section of the **usqlsd.ini** configuration file, select **2**. You are prompted with the current setting for each directive, in turn, which you either retain by just pressing RETURN, or modify as you wish.

Select **5** from the Main Menu to clear the log file **usqlcs.log**. This displays:

```
Log file: /usr/usqls/bin/usqlcs.log
Size of the log file: 130 bytes
Modification date and time: 07/Nov/1997 09:58
Do you want to clear the log file (y/n):
```

This **serv_setup.sh** script can also be used to start (select **3**) or stop (select **4**) the U/SQL Server. These selections simply invoke the **start_serv.sh** and **stop_serv.sh** scripts respectively.

Manually Amending U/SQL Server Directives

The U/SQL Server Directives

The U/SQL Server directives are grouped under three Section Names:

- [Configuration Settings]
This Section Name defines the global settings for the server for all data sources.
- [Data Source Defaults]
This Section Name defines the default directives for all data sources unless overridden in individual data source section(s), below.
- [<Data_source_name>]
This Section Name is the name of a specific data source, for example, <Data_source_name> could be 'books.udd'. There can be any number of data source entries (that is, UDDs) in the settings, describing different applications you may wish to connect to, or different views of the same application.

The following sections detail the directives that can be set under each of the three Section Names in the **usqlsd.ini** configuration file.

- [Configuration Settings] Section
- [Data Source Defaults] Section
- [\[<Data source name>\] Section](#).

Note: *It is very important that the exact spelling, and this includes use of mixed lower and UPPERCASE where appropriate, of both the directives and their values is adhered to.*

Example U/SQL Directives

A sample **usqlsd.ini** configuration text file is:

```
[Configuration Settings]
DefaultServer=centralhost
DefaultPort=7000
NewDictionaryDir=/usr/usqls/bin
LogFileDir=/usr/usqls/bin
LogLevel=0
MsgFileDir=/usr/usqls/bin

[Data Source Defaults]
ReadOnly=Yes
Security=Host
BecomeUser=Y
UnauthorizedAccess=N
SecurityFile=/usr/usqls/security.txt

[Company01]
Dictionary=/u/data/company.udd
Directory=/usr/usqls/bin
Searchlist=/u/data01;/u/data
FileNamePrefix=AA
FileNameSuffix=ZZ
Substitution=##=01;??=xy
TempDir=/u/tmp
```

[Company01] is the Data Source Name (DSN) that you want the ODBC-enabled products to recognize. It is usually the name of the UDD, for example **[company.udd]**, in which case the '.udd' extension is included.

If you are describing multiple companies then each DSN name will be different to the UDD. Refer to the section [Multi-company Support](#).

A minimum **usqlsd.ini** configuration file will usually consist of the following:

```
[Configuration Settings]
DefaultServer=centralhost
DefaultPort=7000
MsgFileDir=/usr/usqls/bin
LogFileDir=/usr/usqls/bin
LogLevel=0

[books.udd]
Directory=../example
```

Transoft U/SQL User Guide

[books32.udd]

Dictionary=../example/books.udd

Directory=../example

Micro Focus COBOL Specific Issues

Micro Focus COBOL runtime

Multiple-tier U/SQL for Micro Focus COBOL assumes that a valid COBOL run-time is installed on the target server system, and that (where necessary) the appropriate License Manager is installed and running.

The COBOL system is accessed by the **COBDIR** environment variable, which needs to contain the full path to the directory where the Micro Focus COBOL product is installed. In addition, because Micro Focus makes use of dynamic linked modules, it is necessary to include the path to the COBOL libraries in the environment variable **LD_LIBRARY_PATH** (some machines use **SHLIB_PATH**). The path to the COBOL libraries is, by default, **\$COBDIR/coblib**, that is the sub-directory 'coblib' of the COBOL directory.

Note: The COBOL **COBDIR** environment variable, must be set prior to starting the U/SQL Server.

To ensure that the **COBDIR** environment variable has been correctly set, run the following:

```
cobrun -V
```

The Micro Focus COBOL copyright message is displayed. If this is not the case, set the **COBDIR** environment variable appropriately.

Refer to your *Micro Focus COBOL System Reference* manual for further information on setting up the COBOL environment.

Using Micro Focus COBOL V3.0/3.1 (UNIX Multiple-tier)

On UNIX platforms, to ensure record locking works correctly with Micro Focus COBOL versions 3.1 or 3.0, the following line should be added to the top of the configuration file which is pointed to by the **COBCONFIG** environment variable:

```
set lock_mode=2
```

If the **COBCONFIG** environment variable is not defined, then the file will be called **cobconfig** and can be found in the location pointed to by the **COBDIR** environment variable. If this file does not exist, simply create it using a standard text editor.

This will ensure that the record locking mechanism used is the same for both COBOL-base applications and U/SQL Adapters, and will not have any adverse effects on your COBOL applications. For more details on the lock_mode run-time configuration, refer to your Micro Focus COBOL documentation.

Changing the lock mode, in the **cobconfig** file, must only be performed when there are no COBOL application programs running. It is recommended that the server is re-booted, before any changes are made, to ensure that this is the case.

Note: This is not required for Micro Focus COBOL version 3.2, and above, or the Single-tier Windows platforms.

Additional Information

Linking external file handlers

Since U/SQL makes use of the standard Micro Focus **EXTFH** file handler, the C-ISAM library can be linked into U/SQL in the same way it would be linked into a Micro Focus COBOL runtime system.

To do this, the following entry in the **makefile** shipped with the U/SQL release:

```
COBLIB= cobol.a
```

must be modified to:

```
COBLIB= cobol.a -m ixfile=cixfile -L/usr/lib +lisam
```

where `/usr/lib` is the UNIX directory containing the C-ISAM library **libisam.a**.

This mechanism is described in detail in the *Callable File Handler* section of the *Micro Focus Reference* manual. This facility does not apply when the standard Micro Focus file system is used.

U/SQL requires the directive **MFIsam=N** to be included in the **usqlsd.ini** configuration file.

Client Configuration

U/SQL Client Installation Files

The installation setup program performs the following tasks:

- It checks that you have the correct version of Microsoft ODBC Driver Manager installed. It is recommended that you have at least, version 2.5 **ODBC32.DLL** located in the **\WINDOWS\SYSTEM** directory. If you do not have this, setup installs it.
- It installs the U/SQL ODBC Driver (**TSODBC32.DLL**) into the **\WINDOWS\SYSTEM** directory.
- It prompts you:
 - What is your server called? Enter the server name, for example, yourserver.
 - What is your server socket port number? Enter the socket port number you noted when you installed the U/SQL Server, for example, 7000.
 - It installs the U/SQL Administrator which is used to set up the [ODBC.INI directives](#) entries.
 - If optionally ordered, it installs the U/SQL Manager, which is used to create, view, amend or delete the UDD for COBOL data sources. It can also be used to set up the **ODBC.INI** directive entries.

Note: *For most data source drivers, only the U/SQL Administrator is available.*

- Optionally it installs a [demonstration application](#), written in Microsoft Visual Basic version 5, for a Books Wholesaler. This demonstration shows examples of various queries, graphics and OLE (Object Linking and Embedding) to Microsoft Excel spreadsheets. The Visual Basic source is provided to help you review the application. The sample data files and corresponding UDD are included so that the application can be run.

For the Books demonstration, it edits the **ODBC.INI** Registry entries to add a section for the **BOOKS32.UDD** data source and dictionary.

- Optionally it installs the Interactive U/SQL utility, [Win USQLi](#), that allows sample queries to be made and provides export/import functions for UDDs. This is a simple ODBC-enabled product that you can use to query the Books Demonstration data or your own application tables. For more details see the [Win USQLi](#) section.
- Optionally it installs the **TestNet** utility that is used to test your network.
- It installs the **License Tool** that is used to install and examine the License. See the [Licensing](#) section.
- It creates a set of U/SQL Client icons in the appropriate Windows program group.

At this point, the installation of the Multiple-tier U/SQL Client software is complete.

Interactive U/SQL Utilities

There are two Interactive U/SQL utilities, one a Windows based application, **Win U/SQLi**, supplied with the U/SQL Client software for both Single and Multiple-tier versions; the other, **usqli**, is supplied with Multiple-tier UNIX U/SQL Server software.

These utilities are not meant to provide comprehensive reporting tools. But they are useful for the development and testing of specific SQL queries. They are also useful for (bulk) INSERTs, UPDATEs or DELETEs to your data. In addition, they provide a facility for exporting and importing UDDs.

Win U/SQLi also provides a means of querying the log file, **usqlcs.log**, and a System and File Information utility. These are described in the sections [How to Query the Log File](#) and [System & File Information Utility](#), respectively.

Details of the usage of the U/SQL utilities can be found in the following two chapters:

- [Win U/SQLi](#)
- [usqli on UNIX Servers](#).

Transaction Processing Syntax

Both the Interactive U/SQL utilities, **Win U/SQLi** and **usqli**, support the Transaction Processing syntax, for selected Data Sources. The syntax is as follows:

| | |
|-----------------------------|--|
| TRANSACTION [MODE] ON OFF | Commence/Terminate transaction processing. |
| COMMIT [WORK] | Commit a transaction. |
| ROLLBACK [WORK] | Rollback a transaction. |

For full details, refer to the [Transaction Processing](#) section.

Win U/SQLi


Win U/SQLi

Win U/SQLi is a Windows based utility, which is used for:

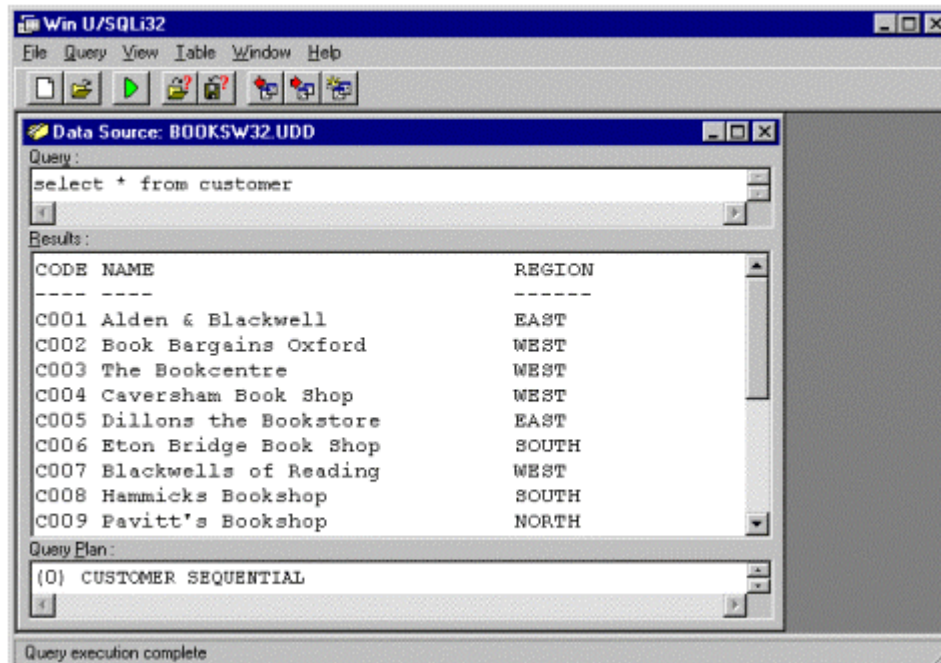
- The development and testing of specific SQL queries.
- Providing (bulk) INSERTs, UPDATEs or DELETEs to your data.
- Providing a facility for importing and exporting UDDs.
- Providing a means of querying the log file, **usqlcs.log**. Refer to the section [How to Query the Log File](#).

To invoke the **Win U/SQLi** utility, select the **Win USQLi** icon from the **Start** menu. The **Win USQLi32** window is displayed:




To open an existing UDD (that is, to connect to an existing UDD) either, click the  (Open) icon or select the **Open UDD** command from the **File** menu. For example, to connect to the Books Demonstration UDD, for Single-tier select **booksw32.udd**, and for Multiple-tier select **books32.udd**.

The Win U/SQLi user interface consists of two windows, a **Query** window and a **Results** window. A third window, **Query Plan**, may be selected by selecting the **Query Plan** option from the **Query** menu.



After entering your query, for example, 'select * from customer' (as shown above), you can execute it by either:

- Clicking the  (Execute the current query) icon
- Pressing CTRL-E
- Selecting the **Execute** command from the **Query** menu.

This displays the Results and the Query Plan, as shown above. Refer to the [The Query Planner](#) section for more information on the Query Planner.

The Win U/SQLi product allows execution of a portion of the SQL specified in the **Query** window by highlighting a section of text and executing the query. This results in only the highlighted text being executed.

The Results set is limited only by the memory available. To view the time taken for the query to execute select the **Performance** command from the **View** menu. The following information is displayed at the end of the Results:

Query Execution & Row Retrieval : 0.71

Total Application Time : 0.749 secs

The Query Execution time is also displayed in the status bar.

Multiple UDDs may be opened simultaneously in separate sets of Query, Results and, optionally, Query Plan windows. A UDD is closed by selecting the **Close UDD** command from the **File** menu.

Note: *In practice, it is not wise to bring back very large results sets because there is noticeable performance degradation due to Win U/SQLi holding all the results data in memory.*

To abort a query at any time, press the ESCAPE key.

The following sections discuss:

- [Win U/SQLi SQL Syntax](#)
- [Win U/SQLi User Login Security](#)
- [Saving and Loading a Query](#)
- [Exporting and Importing UDDs - Upgrading UDDs](#)

- [Win U/SQLi Scripting.](#)


Win U/SQLi

Win U/SQLi is a Windows based utility, which is used for:

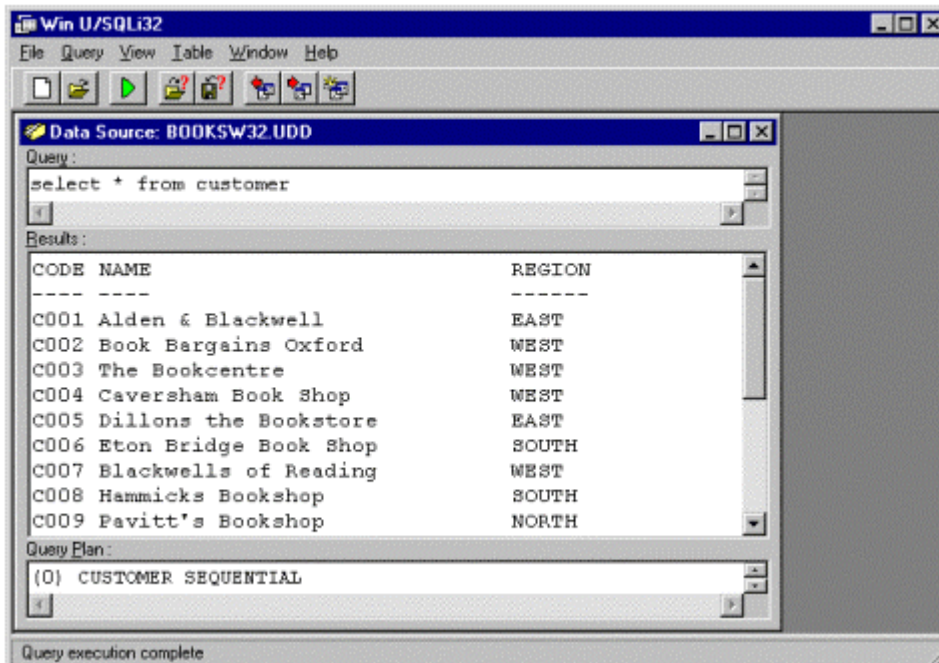
- The development and testing of specific SQL queries.
- Providing (bulk) INSERTs, UPDATEs or DELETEs to your data.
- Providing a facility for importing and exporting UDDs.
- Providing a means of querying the log file, **usqlcs.log**. Refer to the section [How to Query the Log File](#).

To invoke the **Win U/SQLi** utility, select the **Win USQLi** icon from the **Start** menu. The **Win USQLi32** window is displayed:




To open an existing UDD (that is, to connect to an existing UDD) either, click the  (Open) icon or select the **Open UDD** command from the **File** menu. For example, to connect to the Books Demonstration UDD, for Single-tier select **booksw32.udd**, and for Multiple-tier select **books32.udd**.

The Win U/SQLi user interface consists of two windows, a **Query** window and a **Results** window. A third window, **Query Plan**, may be selected by selecting the **Query Plan** option from the **Query** menu.



After entering your query, for example, 'select * from customer' (as shown above), you can execute it by either:

- Clicking the  (Execute the current query) icon
- Pressing CTRL-E
- Selecting the **Execute** command from the **Query** menu.

This displays the Results and the Query Plan, as shown above. Refer to the [The Query Planner](#) section for more information on the Query Planner.

The Win U/SQLi product allows execution of a portion of the SQL specified in the **Query** window by highlighting a section of text and executing the query. This results in only the highlighted text being executed.

The Results set is limited only by the memory available. To view the time taken for the query to execute select the **Performance** command from the **View** menu. The following information is displayed at the end of the Results:

Query Execution & Row Retrieval : 0.71

Total Application Time : 0.749 secs

The Query Execution time is also displayed in the status bar.

Multiple UDDs may be opened simultaneously in separate sets of Query, Results and, optionally, Query Plan windows. A UDD is closed by selecting the **Close UDD** command from the **File** menu.

Note: *In practice, it is not wise to bring back very large results sets because there is noticeable performance degradation due to Win U/SQLi holding all the results data in memory.*

To abort a query at any time, press the ESCAPE key.

The following sections discuss:

- [Win U/SQLi SQL Syntax](#)
- [Win U/SQLi User Login Security](#)
- [Saving and Loading a Query](#)
- [Exporting and Importing UDDs - Upgrading UDDs](#)

- [Win U/SQLi Scripting.](#)

Win U/SQLi SQL Syntax

Win U/SQLi accepts ODBC shorthand SQL syntax in the **Query** window, as defined in the [SQL Syntax Supported](#) section. For example:

```
SELECT * FROM customer WHERE {FN LENGTH(cust_name)} = 17;
```

Note the following:

- Multiple lines can be entered.
- All SQL statements must end with a semicolon ';'.
• The SQL syntax can include, in addition to the **SELECT** statement, **INSERT**, **UPDATE** and **DELETE** verbs.
- **GRANT** and **REVOKE** privileges syntax is supported. See the [Security](#) section.

Note: Multiple **GRANT/REVOKE** commands can be used within one script / execution as long as they are separated by semi-colons ';'.
• The SQL transaction processing syntax (for selected Data Sources) is supported. See [Transaction Processing Syntax](#) section.

- **INFO** <tablename> is supported, printing out the **CREATE TABLE** and **CREATE INDEX** statements for the <tablename> given. This returns information about the table and index definition of the table specified.

Note: Refer to the [SQL Syntax Supported](#) section.

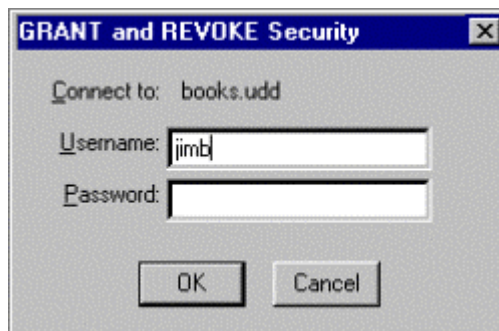
Win U/SQLi User Login Security

Win U/SQLi supports Multiple-tier user connection security and/or GRANT and REVOKE user connection security. For full details on how to set up the security options refer to [Security](#) section.

If Multiple-tier user connection security is in operation then you will be prompted for username and password login entries, as follows:




If GRANT and REVOKE user connection security is also operational, and your username and password are the same as for those for Multiple-tier user connection security, then you will not be prompted for a further login. However, if they are different or only GRANT and REVOKE user connection security is operational, then you are prompted as follows:

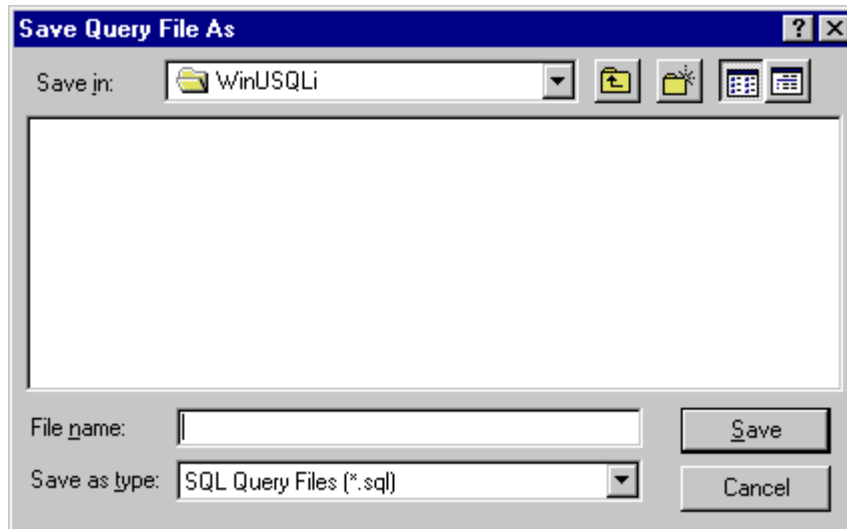


Saving and Loading a Query


Any query in the **Query** window can be saved by either:

- Clicking the  **Save the current query to a file** icon
- Selecting the **Save** command from the **Query** menu.

A **Save Query File As** dialog box is displayed:



This allows you to enter the name you want to give the query.

A saved query may be loaded by either clicking on the  **Load a query file** icon or by selecting the **Load** command from the **Query** menu, which displays the **Load Query** browser. After you select a file the saved query is placed in the **Query** window ready for execution.

Exporting and Importing UDDs - Upgrading UDDs

For COBOL data sources, it may be necessary to upgrade existing dictionaries created with older revisions of U/SQL, to enhance their performance and to take advantage of the increased number of index key parts (from 8 to 64) supported from revision 2.65. This can be achieved by first exporting the existing version of the UDD to a text file representation of the dictionary components and then re-importing these components into a new UDD with the latest structure.

To undertake this you must have Read-Write capability.

Note: For non-COBOL dictionaries, you will already have this textual representation file, called for example, **dictname.ufd**. To create an upgraded UDD, simply repeat the steps detailed in the section [Creating the UDD from the UFD Text File](#). However, it is possible to use the export and import process of Win U/SQLi for any dictionary.

Win U/SQLi can be used to upgrade an existing Single or Multiple-tier UDD as follows:

1. Invoke **Win U/SQLi**.
2. Either click the **Open an existing UDD** icon from the toolbar, or select the **Open UDD** command from the **File** menu, then select the required UDD from the list.
3. Either click the **Export data to a file** icon from the toolbar, or select the **Export** command from the **Table** menu.
4. Select the **UFD Tables** View and the tables you want to export (ordinarily this will be All) by highlighting them, and then clicking the **Add** or **Add All** button.
5. Click **OK**.
6. Provide a File name for the text file, which will hold the exported data. (This file is a textual representation of the UDD).
7. The tables are now exported to the text file in turn. Once a table has been successfully exported, the word 'Completed' is displayed to the right of the table name.
8. Create a new UDD to which the exported text file will be imported, by either clicking on the **Create a new UDD** icon from the toolbar, or selecting the **New UDD** command from the **File** menu. Select where you want the UDD to be created, either Local for Single-tier, or Remote, for Multiple-tier.

Set up the **ODBC.INI** directives, as described, for Single-tier in the [Adding ODBC.INI Entries](#) section, and for Multiple-tier in the [Adding Client ODBC.INI Entries](#) section.

It will be necessary to supply a new name for this UDD file, but it can be renamed to the original UDD file name later.

9. Click the **Import Tables** icon or select the **Import** command from the **Table** menu. Select the text file which was created in step 6 above.
10. An **Import Tables** dialog box is displayed, showing each table as it is imported. Each successfully imported table has 'Imported' displayed to the left of the table name.
11. After all the tables have been imported, click **OK**.
12. Click the **Update selected tables** icon, or select the **Update** command from the **Tables** menu. Select the appropriate data source from the

dialog, and click **Update**. This creates the UDD tables from the UFD tables.

13. The following message is displayed:

All tables successfully updated.

14. Click **OK**.

15. The new UDD file can be renamed to the original name when you are satisfied that it has been created correctly. You can check that the new UDD is correct by performing one or more queries on it using Win U/SQLi.

Note: *If you have activated [GRANT and REVOKE security](#), then there are restrictions on Exporting and Importing.*

Win U/SQLi Scripting

The Windows interactive SQL utility **Win USQLi** can be run in a non-interactive mode which involves sending a series of commands to Win USQLi in the form of a script.

Win USQLi commands are prefixed by 'UDD_' to make them easily distinguishable from SQL commands. However, the script commands can be interspersed with SQL statements. When using a combination of script commands and SQL statements, the script commands must not be terminated by a ';' (semicolon), as are the SQL statements. For example:

```
UDD_CONNECT (DEMO.UDD)
SELECT * FROM CUSTOMER;
```

The basic format of the commands is the command-name followed by comma separated parameters enclosed in parenthesis. For example:

COMMAND (*Parameter1, Parameter2,*)

The commands perform basic tasks such as:

| | |
|--------------------------------|--|
| UDD_CREATE | Creates a data dictionary |
| UDD_CONNECT | Connects to an ODBC data source |
| UDD_IMPORT | Imports into a data dictionary using a UFD file |
| UDD_UPDATE | Updates the data dictionary once the import is completed |
| UDD_DISCONNECT | Disconnects from an ODBC data source |
| UDD_CONFIGURE | Modifies ODBC data source attributes |
| UDD_EXECUTE | Executes an SQL query |
| UDD_EXPORT | Exports the UFD into the specified output file. |

Format and Use of the Win USQLi script file

To execute the Win U/SQLi commands, create a text file which contains the commands, and pass this to the Win USQLi as a run-time command-line parameter. Win U/SQLi script files must have a '.SQL' extension.

For example, consider the script file **c:\tmp\test.sql**, which contains the following Win U/SQLi commands:

```
UDD_CONFIGURE (booksw32.udd, DSD=MF; PATH=f:\usqlc\bookdemo\data\booksw.udd; SRCH=f:\usqlc\bookdemo\data;)
```

```
UDD_CONNECT (booksw32.udd)
```

```
UDD_EXECUTE (SELECT * FROM CUSTOMER, f:\tmp\output.txt)
```

The above script can be executed with the following command:

```
c:\usqlc\winusqli\wusqli.exe c:\data\sql\test.sql
```

Where `wusqli.exe` is the actual file name of the Win U/SQLi executable.

This command can be executed from the Windows Taskbar's **Start** or **Run** option under Windows 95, Windows 98 or Windows NT 4.0.

When the script is executed, it connects to the booksw32 Universal Data Dictionary and issues the SQL query:

```
SELECT * FROM CUSTOMER
```

The result of the query is placed in the output file `c:\tmp\output.txt`. A log file is also produced in the local directory which records each command executed, and its success or failure. The log file has a '.log' extension. For example, if your script file is called **test.SQL** then the log file will be called **test.log**.

Command specification

UDD_CREATE (Environment, Data Source Name, Directives)

The **UDD_CREATE** command is used to physically create a new data dictionary (UDD).

This command uses the following parameters:

| | |
|-------------------------|---|
| Environment | The two values allowed for this parameter are LOCAL or REMOTE, where LOCAL indicates a Single-tier model, and REMOTE indicates a Multiple-tier model. |
| Data Source Name | This parameter is used to specify the name of the data dictionary, for example, 'demo.udd'. This name is used as the data source name 'seen by ODBC-enabled products'. For Multiple-tier the '.udd' extension is mandatory. For Single-tier this extension is optional. However, a full pathname with a '.udd' extension must be specified in full in the PATH directive as described below. |
| Directives | This parameter is used to specify a string containing a list of directives which differ for Single or Multiple-tier. The format of this parameter is a defined directive name, followed by an equals sign ('='), followed by that directives value, and a semicolon(';') delimiter. |

Common directives:

| Directive | Status | Description |
|-----------|----------|---|
| DESC | Optional | Provides a description of the data dictionary for |

| | | |
|--|--|-------------------------|
| | | documentation purposes. |
|--|--|-------------------------|

For example:

DESC=The books Universal Data Dictionary;

Multiple-tier directives:

| Directive | Status | Description |
|-----------|-----------|--|
| SERV | Mandatory | Specifies the server where this dictionary is located. |
| PORT | Mandatory | Specifies the port number used to connect to the server daemon. |
| TIME | Optional | This is a global socket port number time-out specified in seconds, used when retrieving data from the U/SQL Server. If results are not returned within the specified time limit, the connection is assumed to have failed. If the time-out is zero or is not specified, then there is no time-out and the U/SQL ODBC driver will wait indefinitely. |

For example:

PORT=3456;

Single-tier directives:

| Directive | Status | Description |
|-----------|-----------|---|
| DSD | Mandatory | Specifies the data source for this dictionary. The value used must be one of the following: MF - For Micro Focus COBOL data dictionaries ACU - For ACUCOBOL data dictionaries CSM - For C-ISAM data dictionaries IDOL - For IDOL-IV data dictionaries |
| PATH | Mandatory | Specifies the full path and filename of the physical dictionary file. |
| SRCHLIST | Mandatory | Specifies the path or paths of the data files for the dictionary. |
| SUBS | Optional | Specifies the substitution to use. |
| LOGF | Optional | Specifies the directory of the log file used. |
| LOGL | Optional | Specifies the logging level used. |
| PREF | Optional | Specifies the filename prefix to use. |
| SUFF | Optional | Specifies the filename suffix to use. |

For example:

PATH=C:\USQL\DATA\DEMO.UDD;

See the [ODBC.INI Directives](#) section for more details.

An example of how to use the **UDD_CREATE** command is:

```
UDD_CREATE (LOCAL, DEMO.UDD, DSD=MF;
PATH=C:\USQL\DATA\DEMO.UDD;
SRCH=C:\USQL\DATA; LOGL=3;)
```

Note: The directory `c:\USQL\DATA` must exist, otherwise the example program will not work.

UDD_CONFIGURE (Data Source name, Directives)

The **UDD_CONFIGURE** command allows the attributes of an existing data source to be set. The parameters are the name of the existing data source, and then the attributes and values which are to be set or altered. These must be in the same format as specified for the [UDD_CREATE](#) command.

For example:

```
UDD_CONFIGURE (DEMO.UDD, SRCH=C:\USQL\DATA;)
```

UDD_CONNECT (Data Source name)

The **UDD_CONNECT** command establishes a connection to the data source which in the case of U/SQL is usually the name of the UDD. It is this connection that the [UDD_IMPORT](#) and [UDD_UPDATE](#) commands will use for their execution.

For example:

```
UDD_CONNECT (DEMO.UDD)
```

UDD_IMPORT (UFD File)

The **UDD_IMPORT** command imports the specified UFD text file into a Universal Data Dictionary (UDD). You must connect to the UDD using the [UDD_CONNECT](#) command, before issuing the **UDD_IMPORT** command. The UFD file parameter must include the full path and filename of the UFD text file to be used.

For example:

```
UDD_IMPORT (C:\UFD_FILES\DEMO.UFD)
```

UDD_UPDATE ()

The **UDD_UPDATE** command updates the UDD with the imported UFD information. Before issuing this command, you must import the UFD text file into a data dictionary using the [UDD_IMPORT](#) command.

For example:

```
UDD_UPDATE ( )
```

UDD_DISCONNECT ()

The **UDD_DISCONNECT** command closes the connection to the UDD.

For example:

```
UDD_DISCONNECT ( )
```

UDD_EXECUTE (SQL query, Output file)

The UDD_EXECUTE command executes an SQL query using your current UDD connection, and writes the output to a file.

It uses the following parameters:

| | |
|--------------------|--|
| SQL query | A SQL query |
| Output file | The full path and filename of the file to write the output to. |

For example:

```
UDD_EXECUTE (SELECT * FROM CUSTOMER, d:\usqliw\custout.txt)
```

UDD_EXPORT (Output UFD File, [Delimited])

Note: UDD_EXPORT is only available in Win U/SQLi from versions of U/SQL 3.10.407 and onwards.

This command exports the UFD into the specified output file. A second parameter can also be specified as follows:

"SPACE" (the default) - the format of the UFD will be space-padded

"COMMA" - the format of the UFD will be comma-delimited.

These formats are the same as on UNIX with [usqli](#).

For example:

```
UDD_EXPORT (C:\UFD_FILES\NEW.UFD)
```

```
UDD_EXPORT (C:\UFD_FILES\NEWDELIM.UFD, COMMA)
```

Example Script

The following sample script creates a new data dictionary **DEMO.UDD** in the '**C:\USQLCSA\DATA**' directory, connects to it, and then imports the dictionary details from the **TABLES.UFD** text file. It then updates the data dictionary and disconnects from it.

```
UDD_CREATE (LOCAL, DEMO.UDD,
            DSD=MF;
            PATH=C:\USQL\DATA\DEMO.UDD;
            SRCH=C:\USQL\DATA;
            LOGL=3);
UDD_CONNECT (DEMO.UDD)
UDD_IMPORT (C:\SCRIPTS\TABLES.UFD)
UDD_UPDATE ()
UDD_DISCONNECT ()
```


usqli on UNIX Servers

Before you can use the UNIX server based interactive U/SQL utility, **usqli** you must have started the U/SQL Server. This utility allows ODBC shorthand SQL syntax, (as defined in the [SQL Syntax Supported](#) section), to be applied to your data at the server without the need for an ODBC-enabled client product.

This utility is not meant to provide a comprehensive host-based reporting tool. But it is useful for the development and testing of specific SQL queries. It is also useful for (bulk) INSERTs, UPDATEs or DELETEs to your data and for exporting and importing UDD tables. Additionally it can be used to set up GRANT and REVOKE privileges. See the [Security](#) section.

This section discusses the following aspects of the **usqli**:

- [Running usqli](#)
- [usqli command line options](#)
- [usqli User login security](#)
- [usqli facilities](#)
- [Using usqli to query the data definitions within a UDD](#)
- [Using usqli to export and import your UDD.](#)

Running usqli

The **usqli** utility is located in the **bin** directory below the base directory of the U/SQL Server software installation, for instance, **/usr/usqls/bin**. It must be started in the base directory of your data files, for example, **/usr/data_files**, and can be invoked either directly or with the name of the UDD as a parameter. It is a good idea to have the **/usr/usqls/bin**, or your equivalent, in each user's PATH.

The **usqli** utility acts as a client and requires a U/SQL Server process to connect to. On connection, the **usqli** examines the **usqlsd.ini** configuration file to try and locate the server's name and port number. If the entries are not found, **usqli** prompts interactively for either/both the server name or/and port number. The server name and port number entries in the **usqlsd.ini** can be placed in an individual UDD's section, or can be placed in the global [Configuration Settings] section. When trying to connect, **usqli** first looks under the UDD's section for the server information, then under the **[Configuration Settings]** section, and finally prompts the user if not found.

From your data files' base directory, for instance, **/usr/data_files**, either enter:

```
/usr/usqls/bin/usqli
```

or, if the **bin** directory is in PATH:

```
usqli
```

This displays:

```
Interactive U/SQL Utility.
```

```
Copyright (c) Transoft Ltd. 1993-98
```

```
Enter UDD name:
```

Transoft U/SQL User Guide

Type in the UDD name, for example, **books.udd**, and the U/SQL prompt appears:

```
Enter UDD name: books.udd
Connected to server: [Engine v3.00.0002][C-ISAM v3.00.0001]
Opened: 'books.udd'
U/SQL>
```

Alternatively, **usqli** can be invoked with the UDD name as a command line parameter:

```
usqli books.udd
```

which displays:

```
Interactive U/SQL Utility.
Copyright (c) Transoft Ltd. 1993-98
Connected to server: [Engine v3.00.0002][C-ISAM v3.00.0001]
Opened: 'books.udd'
U/SQL>
```

If the UDD entered is not found, the following error message is displayed:

```
*** Error: Cannot open UDD file <name>
ODBC State: C1000
Failed to connect. Aborting...
```

Assuming the UDD is found, most ODBC SQL syntax can be entered, for example:

```
U/SQL> select * from customer;
```

All SQL statements must end with a semicolon ';'.
'

The SQL syntax can include, in addition to the **SELECT** statement, **INSERT**, **UPDATE** and **DELETE** verbs.

Note: Refer to the [Advanced Use of U/SQL Adapters](#) section, for examples of SQL syntax.

Product revision numbering is displayed, for example:

```
[Engine v3.00.0002][C-ISAM v3.00.0001]
```

To exit from the Interactive U/SQL Utility, enter the following (with no semicolon):

```
U/SQL> quit
```

usqli command line options

To obtain the usage of **usqli**, execute the following:

```
usqli -h or usqli -?
```

which displays:

```
Usage: usqli [<options>]
<options>:
```

-c Create dictionary.
 -u Update dictionary.
 -x Do not display Engine and DSD version numbers.
 -i <in_file> Input SQL file.
 -o <out_file> Output file.
 -e <err_file> Output error file.
 -M <uid>[:<pwd>] Multiple-tier Security.
 -G <uid>[:<pwd>] GRANT and REVOKE Security.
 -h|? Display usage.
 <udd_name> Dictionary name.
 <tbl_name>... Table name(s) for '-u'. Must follow <udd_name>.

These command line options are used as follows:

-c Create dictionary <udd_name>.
 This command is used to create an empty UDD. It is usually followed by an import of the UFD text file, representing the structure of your data files, into the new UDD, and then the **-u**, update dictionary command. For example:

```

./usqli -c new.udd    Create the new dictionary
./usqli new.udd      Connect to the new dictionary
and then at the U/SQL> prompt:
  
```

```

U/SQL> import          Import the UFD information
newudd.ufd
U/SQL> quit
  
```

```

./usqli -u new.udd    Update the UDD with the UFD
                      information
  
```

or you can update one table at a time:

```

./usqli -u new.udd CUSTOMER
./usqli -u new.udd TRANSACTION
  
```

Note: You must have Read-Write capability to create UDDs and import into them.

-e <error_file> Direct error messages to <error_file>.
 -i <input_file> Execute the contents of the SQL query file <input_file>.
 -o <output_file> Direct the query results to <output_file>.
 -u <udd_name> Update dictionary <udd_name> for, optionally, table
 [<tbl_name>] <tbl_name>. See above example under -c, 'Create dictionary'.

- x Do not display the U/SQL Engine and Data Source Driver version information. For example: [Engine v3.00.0002][C-ISAM v3.00.0001].
- <udd_name> Connect to the dictionary name <udd_name>.
- M Multiple-tier user connection security. You can supply as command line input your user name, <uid>, and, optionally, your password, <pwd>. Refer to the [Security](#) section.
- G GRANT and REVOKE user connection security. You can supply as command line input your user name, <uid>, and, optionally, your password, <pwd>. If these are the same as the -M <uid> and <pwd>, then you do not have to enter them again with the -G switch. Refer to the [Security](#) section.

usqli User Login Security

The **usqli** utility, supports Multiple-tier user connection security and/or GRANT and REVOKE user connection security. For full details on how to set up the security options refer to the [Security](#) section.

If Multiple-tier user connection security is in operation then you will be prompted for username and password login entries, as follows:

```
./usqli books.udd  
Interactive U/SQL Utility.  
Copyright (c) Transoft Ltd. 1993-98  
Multiple-Tier Security is enabled - login required.  
Enter username (jim): jimb  
Enter password:  
Connected to server: [Engine v3.00.0001][C-ISAM v3.00.0002]  
Opened: 'books.udd'  
U/SQL>
```

Note: *You are prompted with your UNIX login name as the default username.*

If GRANT and REVOKE user connection security is also operational, and your username and password are the same as for those for Multiple-tier user connection security, then you are not prompted for a further login. However, if they are different or only GRANT and REVOKE user connection security is operational, then you are prompted as follows:

```
GRANT and REVOKE Security is enabled - login required.  
Enter username (jim): jimba  
Enter password:  
Connected to server: [Engine v3.00.0001][C-ISAM v3.00.0002]
```

Opened: 'books.udd'

U/SQL>

usqli facilities

If you type 'help' at the U/SQL> prompt, all the commands and their syntax are displayed as follows:

```
U/SQL> help
<SQL statement>;
<filename>[.sql]
! [<unix command>]
export [delimited|delimited2]
export [delimited|delimited2] <table> [<where clause>]
import <filename>
help
rev
info <table>
output [<filename>]
quit
start <filename>[.sql]
tabs
cols
scols
types
stats
set delimited [0|1|2]
showplan [0|1]
```

Some of these commands are for debugging purposes and must only be used if directed to do so by the customer support service. Additional commands may be added in the future. Refer to the server based **README** file to ascertain whether new facilities have been added. See the [Installation & Licensing](#) section for details on displaying or printing the **README** file. Alternatively, refer to the Release Notice in the U/SQL Client software.

Each command is entered at the U/SQL> prompt. Taking each of the commands in turn:

| | |
|-----------------------------------|--|
| <pre><SQL statement>;</pre> | <p>Most SQL syntax can be entered, for example: U/SQL> select * from customer;</p> <p>Note:</p> <ul style="list-style-type: none"> ○ Multiple lines can be entered. ○ All SQL statements must end with a semicolon (;). ○ The SQL syntax can include, in addition to the SELECT statement, INSERT, UPDATE and DELETE verbs. ○ GRANT & REVOKE privileges syntax is supported. ○ The SQL Transaction Processing syntax (for select Data Sources) is supported. See the section Transaction Processing Syntax. |
|-----------------------------------|--|

| | |
|--|--|
| | <ul style="list-style-type: none"> ○ Any output goes to the current setting of 'output'; see the output command below. ○ Refer to the Advanced Use of U/SQL Adapters section, for examples of SQL syntax. |
| <filename> | <p>It is possible to prepare SQL queries in a text file with an '.sql' extension. For instance, newquery.sql could contain the query:</p> <pre>select * from customer;</pre> <p>This query can then be run by entering the file name (without the '.sql' extension):</p> <pre>U/SQL> newquery</pre> <p>The contents of the file is displayed and confirmation to continue is requested before the query is executed.</p> |
| ! <unix command> | Any UNIX command may be executed. |
| export [delimited] export [delimited] <table> [<where clause>] | <p>The export command is normally used with COBOL data sources to export the UDD table(s) as a textual representation. The [delimited] option creates the exported text as quoted comma delimited, that is entries are "quoted" and separated by commas, rather than output in a fixed tabular format. The quoted comma format is much more compact and can be easier to manage for very large dictionaries.</p> <p>Alternatively, you can use the set delimited [0 1 2]command to set delimited on/off, see below. The results of export are output to the current setting of output; see the output command below.</p> <p>The export command can also be used to export the physical data - it is not just internal UDD definition files which may be exported.</p> <p>See the section Using usqli to Export and Import your UDD, below.</p> |
| help | Provides the current list of available commands. |
| import <filename> | The import command is used to import a textual file of UFD components into a UDD. See the section Using usqli to Export and Import your UDD , below. |
| info <table> | Prints the CREATE TABLE and CREATE INDEX statements for the <table> to the current setting of output. |
| output [<filename>] | Changes the current output to the named file or device. If no filename is given all output will be to the screen. If you enter the name of an existing file, you will be asked to confirm that it can be overwritten. |
| quit | Exits usqli. No semi-colon after quit is required. |
| rev | This command displays the revision of the U/SQL Engine and the Data Source Driver. For example: |

| | |
|---------------------------|--|
| | [Engine v3.00.0003][C-ISAM v3.00.0001] |
| set delimited [0 1 2] | This command sets and configures the delimiter for an export command (see export above). 0 uses space-padding, 1 surrounds all data in double quotes (") and separates fields with a comma, while 2 does not quote the data but uses pipe () delimiters. Any of these formats may be imported without setting any delimited (usqli will detect which is being used automatically). |
| showplan <value> | The query plan, generated by the U/SQL Server engine for each SQL statement can be viewed in its simple form by entering "showplan 1". The query plan is displayed prior to query execution. To switch this off again use "showplan 0". Alternatively, if a query plan is desired without actually executing the query, "showplan 5" will switch query execution off but still give back a query plan. This is useful during development of complex queries so that the query plan can be retrieved quickly. An explanation of the query planner is detailed in the The Query Planner section. This facility is 'turned off' by entering showplan 0. |
| start <filename>[.sql] | The commands in the <filename> are executed without prompts. The '.sql' extension is optional. |

Using usqli to query the data definitions within a UDD

Note: The **INFO** command is usually sufficient for most requirements to determine how a table is structured. The following commands are included for completeness, and are mostly present to simulate how an external application would request 'metadata' information.

| | |
|------|---|
| tabs | <p>The tabs command lists tables in the database. It asks for Owner, Table Name, and Table Type in turn. A pattern (including % as a wild card, for example, CUST%R matches CUSTOMER) may be used. It then displays a list of tables which match. Table Type is one of "TABLE" (data table), "SYSTEM TABLE" (internal UDD table) or "VIEW" (read-only view). Entering nothing matches all possibilities, as does entering "%" for any of the categories. To get a list of all data tables, press the RETURN key through both Owner and Table Name, and enter TABLE for Table Type, and in the books database the following is displayed:</p> <pre>U/SQL> tabs Owner: Table Name: Table Type: TABLE TABLE_OWNER TABLE_NAME TABLE_TYPE ----- dba BUDGET TABLE dba CUSTOMER TABLE dba OLINE TABLE dba ORDERS TABLE dba SALEHIST TABLE</pre> |
|------|---|

| | |
|------------------|---|
| | <pre> dba STOCK TABLE 6 records retrieved The tabs command returns the same values as the SQLTables ODBC function call.</pre> |
| <pre>cols</pre> | <p>The cols command lists columns in the database. It asks for Owner, Table Name, and Column Name in turn. A pattern (including % as a wild card, for example, CUST%R matches CUSTOMER) may be used. It then displays a list of matching columns. Entering nothing matches all possibilities, as does entering "%" for any of the items. To get a list of all fields within the CUSTOMER table, for example, enter RETURN for Owner, "CUSTOMER" for Table Name, and RETURN for Column Name. In the books database the following is displayed:</p> <pre> U/SQL> cols Owner: Table Name: CUSTOMER Column Name: TABLE_OWNER TABLE_NAME COLUMN_NAME DATA_TYPE PRECISION ----- ----- ----- ----- dba CUSTOMER CUSTCODE 1 4 dba CUSTOMER CUST_NAME 1 30 dba CUSTOMER CUST_REGION 1 6 3 records retrieved The cols command returns the same values as the SQLColumns ODBC function call.</pre> |
| <pre>scols</pre> | <p>The scols command lists "special columns", which are those which are either unique record identifiers (such as primary keys) or items which are automatically updated whenever a record is amended (U/SQL has none of these). It will ask for Owner, Table Name, and Type in turn. A pattern (including % as a wild card, for example, CUST%R matches CUSTOMER) may be used. The Type must be 1 when connected to a U/SQL database, which is a request for the best unique row identifier for the table (such as the primary key or a row ID field). Entering nothing for Owner or Table Name matches all possibilities, as does entering "%" for either of those items. To find out the best unique identifier for the CUSTOMER table in the books database, enter RETURN for Owner, CUSTOMER for Table Name, and 1 for Type:</p> <pre> Owner: Table Name: CUSTOMER Type: 1 COLUMN_NAME DATA_TYPE PRECISION LENGTH SCALE ----- ----- ----- CUSTCODE 1 4 4 0 1 records retrieved The scols command returns the same values as the</pre> |

| | SQLSpecialColumns ODBC function call. | | | | | | | | | | | | | | | | | | |
|------------|--|----------------|------------|--------------|-------------|--------------|-------------|----------|---|----------------|---|-------|----------|----------|------|------|---|------|------|
| types | <p>The types command lists all the possible data types supported by U/SQL. It prompts for a value. If the value is given, the type details matching that value are listed, otherwise all types are listed. For example:</p> <pre>U/SQL> types</pre> <p>Type:</p> <table border="1"> <thead> <tr> <th>TYPE_NAME</th> <th>DATA_TYPE</th> </tr> </thead> <tbody> <tr> <td>CHAR</td> <td>1</td> </tr> <tr> <td>NUMERIC</td> <td>2</td> </tr> <tr> <td>DECIMAL</td> <td>3</td> </tr> <tr> <td>LONG</td> <td>4</td> </tr> <tr> <td>SHORT</td> <td>5</td> </tr> <tr> <td>DOUBLE</td> <td>8</td> </tr> <tr> <td>DATE</td> <td>9</td> </tr> </tbody> </table> <p>7 records retrieved</p> <p>The types command returns a summary of the information of the SQLGetTypeInfo ODBC function call.</p> | TYPE_NAME | DATA_TYPE | CHAR | 1 | NUMERIC | 2 | DECIMAL | 3 | LONG | 4 | SHORT | 5 | DOUBLE | 8 | DATE | 9 | | |
| TYPE_NAME | DATA_TYPE | | | | | | | | | | | | | | | | | | |
| CHAR | 1 | | | | | | | | | | | | | | | | | | |
| NUMERIC | 2 | | | | | | | | | | | | | | | | | | |
| DECIMAL | 3 | | | | | | | | | | | | | | | | | | |
| LONG | 4 | | | | | | | | | | | | | | | | | | |
| SHORT | 5 | | | | | | | | | | | | | | | | | | |
| DOUBLE | 8 | | | | | | | | | | | | | | | | | | |
| DATE | 9 | | | | | | | | | | | | | | | | | | |
| stats | <p>The stats command lists tables and indices in the database. It asks for Owner and Table Name. A pattern (including % as a wild card, for example, CUST%R matches CUSTOMER) may be used. It then displays a list of tables and index-parts which match. Entering nothing matches all possibilities, as does entering "%" for any of the items. To get a list of the indices within the CUSTOMER table, for example, enter RETURN for Owner and "CUSTOMER" for Table Name. In the books database the following is displayed:</p> <pre>Owner: Table Name: CUSTOMER</pre> <table border="1"> <thead> <tr> <th>TABLE_NAME</th> <th>NON_UNIQUE</th> <th>INDEX_NAME</th> <th>TYPE</th> <th>SEQ_IN_INDEX</th> <th>COLUMN_NAME</th> </tr> </thead> <tbody> <tr> <td>CUSTOMER</td> <td>0</td> <td>CUSTOMER_IX001</td> <td>3</td> <td>1</td> <td>CUSTCODE</td> </tr> <tr> <td>CUSTOMER</td> <td>NULL</td> <td>NULL</td> <td>0</td> <td>NULL</td> <td>NULL</td> </tr> </tbody> </table> <p>2 records retrieved</p> <p>This lists the basic table as having 1000 rows, and there being a single index, called CUSTOMER_IX001, which is unique and contains the field CUSTCODE. This command returns the same values as the SQLStatistics ODBC function call.</p> | TABLE_NAME | NON_UNIQUE | INDEX_NAME | TYPE | SEQ_IN_INDEX | COLUMN_NAME | CUSTOMER | 0 | CUSTOMER_IX001 | 3 | 1 | CUSTCODE | CUSTOMER | NULL | NULL | 0 | NULL | NULL |
| TABLE_NAME | NON_UNIQUE | INDEX_NAME | TYPE | SEQ_IN_INDEX | COLUMN_NAME | | | | | | | | | | | | | | |
| CUSTOMER | 0 | CUSTOMER_IX001 | 3 | 1 | CUSTCODE | | | | | | | | | | | | | | |
| CUSTOMER | NULL | NULL | 0 | NULL | NULL | | | | | | | | | | | | | | |

Using usqli to export and import your UDD

The usqli **export** command is normally used with COBOL data sources to export the UDD to a text file. See the section [Exporting and Importing your UDD](#).

You can selectively export as the full **usqli** command implies:

```
export or export <table> [<where clause>]
```

The usqli **import** command is used to import a text file of UFD components into a UDD. See the [Modifying a UDD](#) section.

Note: *For both COBOL and non-COBOL dictionaries, it is possible to use the export facility of **usqli**.*

If you have activated [GRANT and REVOKE security](#), then there are restrictions on Exporting and Importing.

JDBC Client

JDBC Driver support

U/SQL 3.10.400 and above provide support for the **JDBC Driver**. To use the JDBC driver, take the following steps:

1. The JDBC driver **td.jar** is located in the **jdbc** subdirectory of the root directory of the U/SQL Client/Server Installation CD. Copy this file into a directory of your choice, for example, **c:/my_directory**.
2. Include the full pathname of the JDBC driver, for example, **c:/my_directory/td.jar** in the classpath.
3. The utility **jusql.jar** is also located in the **jdbc** subdirectory of the root directory of the U/SQL Installation CD. Copy this file into a directory of your choice, for example, **c:/my_directory2**.
4. Include the full pathname of the **jusql.jar** utility, for example, **c:/my_directory2/jusql.jar** in the classpath.
5. Run "**java jusql**".

URLs take the form:

"jdbc:transoft://server_name:port/udd_name"

For example:

"jdbc:transoft://sco_os:7000/books.udd".

The driver name is **jdbc.transoft.Driver**.

INI Directives

ODBC.INI Directives

The **ODBC.INI** directives determine how Single-tier U/SQL operates.

When Windows applications access a specific data source through ODBC, they do so by making calls to the ODBC API. This is implemented by Microsoft as a Dynamic Linked Library (DLL) and referred to as the Microsoft ODBC Driver Manager. The **ODBC32.DLL**, after performing some basic parameter and function sequence checking, then redirects the ODBC function call to the appropriate ODBC driver, Single-tier U/SQL **TSENG32.DLL**.

The Driver Manager determines the name and location of the correct ODBC Driver from the Data Source Name (DSN) given by the ODBC-enabled application and from information in the **ODBC.INI** directive settings.

Other directives control and configure the U/SQL system to interface better with the client application(s) being used.

Where the ODBC.INI Directives Reside

On Windows 95/98 and Windows NT or Windows 2000 the *ODBC.INI* directives are placed in the Windows Registry folder **HKEY_CURRENT_USER \ Software \ ODBC \ ODBC.INI**. This folder contains subfolders for each Section Name containing the relevant directive settings.

Note: *It is recommended that you do not use the Registry editor (**regedit** for Windows 95/98; **regedt32** for Windows NT and Windows 2000), to add or modify entries as it is possible to make mistakes; use the [U/SQL Administrator](#) instead.*

Some newer applications look for System data sources (requiring Microsoft's ODBC 3.0 or later to be installed) rather than User data sources. System data source are 'seen' by all users accessing the machine, while User data sources are only 'seen' by the user who created them. In general, if a System data source has been setup any application should be able to use this data source. However, you may need to set up both System and User data sources for differing ODBC-enabled applications you are running.

A System data source is separately set up and configured in exactly the same way as a User data source, except that its directives are held in a different part of the Registry. Whereas the User data sources are held in the folder 'HKEY_CURRENT_USER', as detailed above, the System data sources are held in 'HKEY_LOCAL_MACHINE' folder. Refer to the section [System & User Data Sources](#).

ODBC.INI Directives

U/SQL requires certain **ODBC.INI** directive settings that include:

- The name of the data source you want to connect to which, in the case of U/SQL, is usually the name of the UDD.
- The name of the ODBC driver to use which, for Single-tier U/SQL, is **TSENG32.DLL**.

- The name of the Data Source Driver(s), for ACUCOBOL this is **TSACU32.DLL** and Micro Focus COBOL this is **TSMF32.DLL**.
- The name and path of the UDD.
- Other optional entries.

There are three Section Names containing the directives:

- [Transoft U/SQL Configuration]

This Section Name defines the global settings for all data sources.

- [Transoft U/SQL Defaults]

This Section Name defines the default directives for all data sources unless overridden in individual data source section(s), below.

- [<Data_source_name>]

This Section Name is the name of a specific data source, for example, <Data_source_name> could be '**booksw32.udd**' (the '.udd' extension is not mandatory). There can be any number of data source entries in **ODBC.INI** settings, describing different applications you may wish to connect to, or different views of the same application.

The following sections detail the directives that can be set in each of the three Section Names in the **ODBC.INI** Registry folders:

- [\[Transoft U/SQL Configuration\] Section](#)
- [\[Transoft U/SQL Defaults\] Section](#)
- [\[<Data_source_name>\] Section](#).

UNIX Client-Side Directives

When using a UNIX client, either a program or [usqli](#), it is necessary to provide information about the server and port the server would be running on. usqli uses the file **usqlsd.ini**, and looks up DefaultServer= and DefaultPort=, or the Server= and Port= lines from the data source section if this is provided.

For example:

```
usqlsd.ini
[configuration settings]
defaultserver=otherserver
port=7000
[books.udd]
server=myserver
port=7001
```

Any attempt to connect to `books.udd` would go to `myserver` on port 7001, whereas any other UDD would be looked for on `otherserver` on port 7000.

For programs, this information is instead contained within a file called **ODBC.INI**, which only has UDD names (no "configuration settings" section) so all UDDs being connected to must be listed within this file. It is permissible to change the name of this file by setting the environment variable **ODBC_INI** to any file name required.

Setting up this file is not necessary on Windows environments because the ODBC Administrator stores all the server and port information.

Configuration Section

In Single-Tier systems there is no **Configuration Section** of INI directives.

For Multi-Tier systems on Windows, the **Configuration Section** is amended from within the [U/SQL Service Manager](#) by selecting the **Configuration Settings** tree element in the left-hand pane under the chosen server.

For Multi-Tier systems on UNIX, either use the **serv_setup.sh** script or edit the **usqlsd.ini** file directly and edit settings under the **[Configuration Settings]** header.

The **Configuration Section** defines the global settings for all data sources. The section can contain any or all of the following directives:

| | |
|--------------------|--|
| NewDictionaryDir | This defines the directory where new dictionaries will be created. The default is the directory where the software was installed. For example: NewDictionaryDir=C:\Program Files\USQLC |
| LogFileDir=C:\TEMP | The directory path where the log file will be created. This default entry is automatically created at the time of installation. The log file is USQLCS.LOG . |
| LogLevel=0 | This defines the message log level. This default entry (0) is automatically created at the time of installation. See the USQLCS.LOG File and Log Levels section. |
| MsgFileDir | This defines the path to the error message file USQLCS.MSG . This default entry is automatically created at the time of installation. For example: MsgFileDir=C:\WINDOWS\SYSTEM\ |
| ODBCVer=01.00 | U/SQL Revision 3.00 is ODBC Revision 2 compliant. This directive can be set to inform any ODBC-enabled product that U/SQL is an ODBC Revision 1 Driver. Note: <i>Only use this setting if you wish the ODBC-enabled product to restrict its call interface to the ODBC Revision 1 specification.</i> |
| CacheNumPages=n | This optional directive determines the size of the buffer pool cache (in pages) that manages the UDD. The default is 256 pages. If you have a large UDD there may be benefit in increasing this value. |
| CachePageSize=n | This optional directive determines the maximum size of the row that can be created in any temporary table, for example, due to the need to sort the results set. The value of 'n' is specified in K |

| | |
|--|---|
| | (1024) bytes, and has a default value of 2 and the minimum is 2. It should only be modified if you receive the error message "Table row length exceeds cache page size". |
| Num2Char=R/L | Determines is a number is to be stored in a string variable as [R]ight or [L]eft (the default) justified. This directive can also be specified in the [Data Source Defaults] section, or under the DSN. |
| TempTablePages=n | If U/SQL runs out of temporary file space this can be increased to allow more. The default is 512. |
| FHUserName, FHPassword, FHREDIR | Override values to allow multiple fileshares and fileshare security for Micro Focus Net Express. |
| Licence=<filename> or License=<filename> | Sets the location of the license file. This is automatically entered when the licence utility is run, so it must not be amended. |
| MaxDSD=n | UNIX only. Allows more than the default of 250 DSNs to be set up within the usqlsd.ini file. |
| ConversionCheck | Defaults to Y. If set to N conversion errors appear only as warnings in the log file, rather than errors. |
| NumericDefaultDouble | COBOL only. Defaults to N. If set to Y decimal-type numerics become "double" type in U/SQL rather than "numeric" - this can help MS Access to read the data if using a "," as the decimal separator. |
| CHECKPK | Micro Focus COBOL and U/FOS only. Defaults to Y. If set to N then U/SQL will allow updating of the primary key value. Note: <i>On many systems this will fail at the Micro Focus COBOL level, so it must not be changed unless testing has been carried out to prove that the version of Micro Focus COBOL being used supports amendment of primary key values.</i> |
| DefaultServer | UNIX Only. Used only by usqli . |
| DefaultPort | UNIX Only. Used only by usqli . |
| MFISAM | Micro Focus COBOL only. Some versions of Micro Focus COBOL do not support op code 6 (file enquiry). If set to N this directive prevents U/SQL |

| | |
|----------------------|---|
| | from using this call to determine the physical file structure. Defaults to Y. |
| TransLogDir | C-ISAM only. Must be set in order for transaction processing to operate. |
| UDDVER | If set to 2, creation of a new UDD will generate a type-2 UDD, which has a second level of index (the start of each page of data appears in a separate index). This speeds up access in large UDD's. Recommended only when hundreds of tables are present within a single UDD. Set to 1 by default. |
| SubSeparator | By default the substitution list data item is separated by ";" or ":", or "," on UNIX. However, if any of those characters are to be used in a pathname, it is necessary to amend the substitution list separator accordingly. This directive must be set to a series of characters which will be treated as separators, either using the literal character or its hex value. For example, to set "!" and ";" to be separators, set SubSeparator=;!, or SubSeparator=0x3b0x21 (! is hex 0x21 and ; is hex 0x3b). |
| COMPSTORAGE | U/FOS Only. Allows amendment to the way COMP values are stored. See the U/FOS Reference Manual for details. |
| ShowLogicallyDeleted | U/FOS Only. Defaults to N. Display records marked as logically deleted as if they were 'live'. |
| SetOptionWarn | Defaults to N. If set to Y any errors produced by client programs requesting SQLGetInfo options that are not supported by U/SQL will generate a warning and return success to the calling application, rather than the default error. |

Data Source Defaults Section

The **Data Source Defaults** Section defines the default directives for all data sources unless overridden in the individual data source section(s).

In Single-Tier systems there is no **Data Source Defaults** Section of INI directives.

For Multi-Tier systems on Windows, the **Data Source Defaults** Section is amended from within the [U/SQL Service Manager](#) by selecting the **Data Source Defaults** tree element in the left-hand pane under the chosen server.

For Multi-Tier systems on UNIX, either use the **serv_setup.sh** script or edit the **usqlsd.ini** file directly and edit settings under the **[Data Source Defaults]** header.

This section contains default values, which can be overridden by setting the directive within the actual data source. This section can contain any or all of the following directives:

| | |
|------------------------|---|
| Dictionary | <p>This is the name and path of the UDD you defined when you set it up. For the Books Demonstration, the dictionary is BOOKSW.UDD in the path shown. The '.UDD' extension is mandatory. For example:</p> <pre>Dictionary=C:\Program Files\BOOKDEMO\DATA\BOOKSW.UDD</pre> |
| Directory | <p>This optional entry defines the U/SQL Server's working directory. The default is as above.</p> <p>If this directive is not specified, then the working directory is the directory where the UDD is located.</p> <pre>Directory=C:\Program Files\USQLC</pre> |
| OpenExclusive={Yes No} | <p>This optional entry is for Micro Focus COBOL only. If set to 'Yes', it ensures that all files are opened exclusively which is likely to improve I/O performance. However, care must be taken in its use. If you have multiple records types in the same physical file then U/SQL may need to open the same file more than once, and with OpenExclusive set to 'Yes', you get the error message:</p> <pre>Error Code: 54 - Requested file locked.</pre> <p>Note: You must ensure that when creating or modifying a UDD using the U/SQL Manager that OpenExclusive is NOT set to 'Yes'.</p> |

| | |
|--------------------------|---|
| <p>ReadOnly={Yes No}</p> | <p>This optional entry, set to 'Yes' if you wish no INSERTs, UPDATEs or DELETEs to be allowed on a global basis to the data files.</p> <p>If an attempt is made to INSERT, UPDATE or DELETE read-only data sources, the following message is displayed:</p> <p>Error - Attempt to update read-only database.</p> <p>Note: <i>Read-only can also be set by licensing.</i></p> |
| <p>SearchList</p> | <p>This is an optional entry. Each directory in the list is separated by the default separation characters (';', ':' or ',' on UNIX, ';' or ':' on Windows) unless overridden by the SubSeparator directive. For each file to be opened, an attempt is made first in the current working directory, which is the directory the UDD is located in unless changed by the Directory= directive (see above). If the file does not exist there, an attempt to open the file will be made in each directory in the search list in turn, until the file is found or the search list is exhausted.</p> <p>When creating your UDD, you can specify each physical file either as just its file name to be found in one of the searchlist directories, or with an absolute path. The former option is preferable as it provides greater flexibility.</p> <p>When using COBOL dictionaries, in order to search along paths specified by the SearchList entry for a given file, you must ensure that there is no directory path specified within the UDD for that table. You can check this by loading the U/SQL Manager, opening the required dictionary and table, and examining the Directory entry field. If a path exists in this field, simply delete it and close the table.</p> <p>Instead of using the U/SQL Manager, to ensure all tables have no individual paths specified, perform the following SQL statement in the Interactive U/SQL utility, Win U/SQLi:</p> <pre>update cobol_table set pathid=0;</pre> <p>Example setting:</p> |

| | |
|---|---|
| | SearchList=C:\DATA\;F:\NEWDATA\ |
| <p>FileNamePrefix=AA FileNameSuffix=ZZ</p> | <p>These are optional entries. The Prefix and Suffix entries, if present, are added to the beginning and end respectively of ALL physical file names contained in the UDD. For example, suppose there is a file in the UDD called CUST, then the physical file that U/SQL will attempt to open will be, with the above Prefix and Suffix, AACUSTZZ.</p> |
| <p>Substitution=??=XY;##=01</p> | <p>This is an optional entry and is used to distinguish between multiple companies using the same UDD; refer to the Multi-company Support section.</p> <p>Each substitution in the list is separated by a semi-colon ';'. These substitutions are applied, in the order listed, to the full pathname of a file before the file is opened. This applies to any path specified for a file including those defined by the Searchlist directive.</p> <p>Each element in the substitution list is of the form string1=string2 where string1 is the text to be replaced (before) and string2 is its replacement (after). For example, if a filename is stored in the data dictionary as "??LEDGER.##" and the substitution entry is as above, the filename to be opened would be "XYLEDGER.01".</p> <p>The length of the text to be replaced (before) does not have to be the same length as the replacement string (after).</p> <p>Note: <i>FileNamePrefix</i> and <i>FileNameSuffix</i> are applied to the file name before any Substitutions. Thus, <i>FileNamePrefix</i> and <i>FileNameSuffix</i> can include substitution characters.</p> |
| <p>FixedDateOffset=nn</p> | <p>This optional directive allows you to define a 'cut-off' year below which dates with two digit years are considered to be 20nn. The value of 'nn' may be 0 to 99. Assume, nn=30, then any year 0 to 29 is considered 2000 to 2029 and any year 30 to 99 is considered to be 1930 to 1999.</p> <p>Before using this directive you should check that existing date data does not come into the range prior to the offset date. Obviously, setting the directive will</p> |

| | |
|-------------------------------|--|
| | <p>'move' any such dates forward by 100 years! To check whether any records exist with dates in this range issue the following query:</p> <pre>select count(*) from <table> where <date> <"19nn-1-1"</pre> <p>If the count is zero then no such records exist.</p> <p>Note: <i>Illegal dates are treated as NULL and a warning is written to the log file.</i></p> |
| TempDir=C:\TEMP | <p>This is an optional entry that specifies the directory in which any temporary files are opened. The default is the current working directory.</p> <p>The following optional directives specify foreign character set support. Refer to the section <i>Foreign Character Set Support</i> and the text file, tstrans.dat, found in the U/SQL installation directory, for example, C:\Program Files\USQLC.</p> |
| TranslationFile | <p>Set this directive to enable translation on UNIX. For example:</p> <pre>TranslationFile=tstrans.dat</pre> <p>For Windows systems this must be set using the U/SQL Administrator for Single-tier or the U/SQL Service Manager for Multiple-tier. See Foreign Character Support for details.</p> |
| HideSystemTables=Y | <p>Even when a third-party application asks for system tables to be returned do not return them. The default is N.</p> |
| TrueSFU=Y | <p>Allow multiple locks within Select-For-Update statements. The default is N.</p> |
| StripUnprintable=Y | <p>If a single unprintable character is returned as part of a C-ISAM field this directive assumes the field is blank. The default is N.</p> |
| ACUConfigFile=<path>/USQL.CNF | <p>This optional entry is for ACUCOBOL DSD only. It allows you to apply a 'unique' set of ACU configuration settings specific to the U/SQL server process. An example configuration file, USQL.CNF</p> |

| | |
|--|--|
| | <p>V-STRIP-DOT-EXTENSION 0</p> <ul style="list-style-type: none">• The normal performance of U/SQL will not be affected if the directive is not set• This USQL ACU configuration file may contain ACU settings that affect the functioning of the file handler• The USQL-ACU configuration file does not mirror the ACU configuration file typically used by a COBOL application.• This new file should only contain the variables required by USQL to load the ACU file handler (a selective approach is therefore advised when populating your U/SQL ACU configuration file). <p>Configurable variables that can be set are defined in the ACUCobol Vision FileHandler API documentation.</p> |
|--|--|

Data Source Section

In Single-tier systems set the **Data Source Section** through the **Advanced** tab of the [U/SQL Administrator](#).

For Multiple-tier systems on Windows, the **Data Source Section** is amended from within the [U/SQL Service Manager](#) by selecting the data source name tree element in the left-hand pane under the chosen server.

For Multiple-tier systems on UNIX, either use the **serv_setup.sh** script or edit the **usqlsd.ini** file directly and edit settings under the data source name (**[<data_source_name>.udd]**) header.

This section may contain any of the entries described in the [Data Source Defaults Section](#), and they will override the default values.

Foreign Character Set Support

U/SQL supports foreign character set mapping via a map file. This map file is set using the [U/SQL Administrator](#) for Single-tier systems, the [U/SQL Service Manager](#) for Multiple-tier Windows systems, or via the **usqlsd.ini** file for UNIX Multiple-tier systems. See the directives section for how to set the [TranslationFile](#) directive.

The **tstrans.dat** file which is installed with U/SQL contains the default mapping, and may be amended as detailed in the comments contained within the file.

You can browse for the name of your translation table file when the translation facility is selected within the ODBC Setup when using the U/SQL Administrator. The default **TSTRANS.DAT** translation table file is supplied in the **WINDOWS\SYSTEM** directory. This skeleton file needs to be modified to satisfy your particular foreign character set requirements.

In the U/SQL Client software installation directory there are example translation table files for German, Danish and Swedish, and others will follow, which are identified by having a '.TRN' extension.

Note:

- *All data, including SQL statements, are filtered through the translation table file. Hence, special care must be taken not to map characters in the A-Z, a-z, 0-9 range or some of the special characters, such as '{' and '}', since this may cause queries to fail. For example, if S is mapped to X, all **SELECT...** statements would fail and the '{' and '}' characters are used as SQL escape characters, in cases such as ...{oj.....}.*
- *An inverse translation table is automatically setup to ensure the correct mapping takes place for INSERTs and UPDATEs.*
- *The translation and inverse translation tables are printed to the log file if the directive LogLevel=4, see the [The USQLCS.LOG File and Log Levels](#) section.*

Advanced Directives

| | |
|----------------------|--|
| AllowInvalidDates=N | Do not check that a date has valid values, but let through such dates as 00-00-0000 or equivalents. |
| AlternativeIndex=N | If set to Y this enables overlapping index-use. This is done for C-ISAM and COBOL at the update dictionary stage, when a file UDDALTIND is populated. (If not present, recreate the dictionary to enable it). During update all indices are examined to see if there are overlapping fields which could be used to map onto the existing index. For example, if in CUSTOMER a field CUSTSHORT was created at offset 0 with length 2, it could be used as a key field by mapping it onto CUSTCODE. Any number of fields can be used to overlay the index as long as they are contiguous - when an index is split into sections around the record this option will not pick them up as valid since the logic of reassembling the key field would become too complex. |
| AutoCorrectIndices=Y | Enables MF, ACU and C-ISAM to fix the order of indices behind the scenes, matching on the basis of field offsets and lengths. Note that if an index cannot be matched an error is generated. Any amendment to order is logged as a DBG message in the logfile. This will not affect the index order in an INFO command, nor will the index number reported in a query plan be affected. |
| AutoTransProc=N | Activates transaction processing. NOTE must be set to Y if transactions are to be used. Also must not be set in Micro Focus COBOL when Fileshare is not running. |
| BaseCalcDate=0 | If set to a value, that value becomes the base date for Julian date calculations within column expressions, thus enabling complex date calculations. Note that for the case where the customer was using 999999-DATE, the Julian value was set to 584388. |
| BBCRAMDGBB=" , - ." | For BB, changes the compression characters for DG compression methods. |

| | |
|-------------------|---|
| BBCRAMSDOS=" /,." | For BB, changes the compression characters for SDOS compression methods. |
| BBGLOBLOCK | Sets the file name to be used for a list of global files, that is, ones where the BBPREFIX does not apply. |
| BBJULIANFLAGS | The BBASIC ISAM DSD uses the same Julian date libraries as the Universal Business Language (U/BL) interpreter. The INI directive BBJULIANFLAGS has been introduced into the BBASIC ISAM DSD to reflect the same functionality that can now be found in U/BL Rev 2.10. |
| BBLASTKEY | This directive controls the way new duplicate records are added to duplicate indices in the ALPHA Release BB ISAM DSD. If set to Y , duplicate keys are added after the last entry otherwise they are added before the first entry. |
| BBLOCKNAME | This sets the name used for the lock. 0 (default) is the logical file name (Ifname from BB_TABLE). 1 is the table name (tabname from BB_TABLE). 2 is the physical file name (dbname from BB_LOGFILE). |
| BBLOCKTYPE | This sets the locking method used for locking BB ISAM records. If BBLARGEFILES is used, this is always set to UBL regardless of the INI directive. |
| BBPREFIX | This sets the prefix for lock names within BB. |
| BecomeUser=N | If set to Y , the U/SQL Server changes its user-ID and group-ID to that of the user. This ensures that operating system file permissions take effect. If set to N , the username and password are still validated, but the U/SQL Server does not change user. |
| CacheTables=N | When set to Y , the UDD tables structures are stored on the connection handle between statements, thus speeding up large numbers of small queries on the same table within a single connection. For example, multiple inserts on the same file. |

| | |
|-------------------|---|
| ConversionCheck=Y | If set to Y , any data conversion errors (such as numeric overflow when converting an integer into a smallint) will generate ODBC errors. If set to N a message will be written to the log file, but processing will continue. |
| CopyParams | Set to Y to cause parameter buffers to be reallocated. When SQLBindParameter is called to set a parameter value the calling app normally allocates the buffer and passes it to our driver. Setting this directive to Y causes us to allocate a new buffer and copy the value there. This was introduced because Powerbuilder was writing over the original buffer causing the parameter to be invalid during the fetch. |
| CreateHJEUDD=N | If set to Y , any new UDD created will include the qualifier field in UDDTABLES and will include the UDDEXTSRC table, and will be enabled (assuming licensing requirements are met) for the HJE. |
| DecimalRound=N | Rounds decimal numbers in C-ISAM, so 69.599999 would become 69.60 if loaded into a field with ndec=2. |
| DefaultOwner | In the HJE, this directive controls the default owner field to be used when no qualifier is entered, that is, when a table is referenced just by the table name. So, " select * from customer " would be amended to " select * from DefaultQualifier.DefaultOwner.customer ". |
| DefaultQualifier | In the HJE, this directive controls the default qualifier field to be used when no qualifier is entered, that is, when a table is referenced just by the table name, or when only the owner is given. So, " select * from customer " would be amended to " select * from DefaultQualifier.DefaultOwner.customer ". Also, " select * from owner.customer " would become " select * from DefaultQualifier.owner.customer ". |
| DefaultRows=1000 | Specifies default number of rows if nrows is not set in UDDTABLES |

| | |
|-----------------------|---|
| DisableAPI= | Specify the name of an ODBC API function or a semi-colon delimited list of functions to disable. When disabled SQLGetFunctions will say that this particular API function or functions is not supported by our driver. This was introduced to allow a customer to get his application working in the same way as with an older version of our driver. The problem was that we introduced the function SQLDescribeParam , which we previously didn't support. Now it is available, Powerbuilder uses a completely different method of retrieving data and this highlighted various problems. |
| DistinctJump=Y | When performing a query such as " select distinct year from salehist ", the index for salehist will be used and once the first year-value is obtained, the cursor will jump directly to the next year value, not scan through the file. |
| DivZeroRtn=0 | |
| EnableAPI= | Some ODBC API functions are disabled by default (Just SQLMoreResults so far). Use this directive to enable them. See DisableAPI for the correct syntax. |
| Fixed6Char | Allows setting of the 'default' character with C-ISAM fixed6char fields. This character, defined as a two-character hex value (20 for space) will be used instead of zeroes. |
| FixedLengthCharType=N | If set to Y , all CHAR-fields will be padded with spaces. Without this, all CHAR-type fields will be space-stripped. |
| ForceSimilarIndex=N | If set to Y , the following condition causes a change of index to be used for the query plan: If two tables exist in the query T1 and T2 such that T2 is using an index and T1 is scanning sequentially, and T2 is using a variable from T1 to read, and this variable being used forms part of an index in T1, then change to read along that index. This is designed to speed up reads so that the rows for T2 will be, approximately at least, fed to it in the correct order. |
| HJEShowplan=N | If set to Y , the HJE does not open connections until execute-time, allowing query plans to be |

| | |
|----------------|--|
| | produced without requiring all the remote data sources to be set up. |
| IndexStats=N | <p>If set to Y, Indexstats allows use of UDDSTATS to determine choice of index. If UDDSTATS has not been set up, this will have no effect. If UDDSTATS has been populated, information there will be used to determine costing information for each index and part thereof. For example, if a table has 1000 rows and an index ABC, and there are 500 distinct AB's, then having AB will be costed as returning, on average, 2 rows (1000/500).</p> <p>Can also be set to P, meaning partial, and A for All. These differ from N and Y respectively in that they turn on the new index costing algorithm. Thus:</p> <p>N - use old method and no stats P - use new method and no stats Y - use stats when available or old method A - use stats when available or new method.</p> <p>The new index analysis method takes into account ORDER BY and DISTINCT statements, and is simpler and (hopefully) more accurate.</p> |
| LeadingZero=Y | If set to Y , leave leading 0 on a decimal 0.05 number. If set to N , remove the 0. |
| LocalQualifier | In the HJE, this directive allows explicit reference to a local table. If a table is referenced with this qualifier, the table will be regarded as local, so " select * from LocalQualifier.owner.customer " would be the equivalent of " select * from owner.customer ". Note that with local tables the owner field is ignored, so a local qualifier is all that is required. |
| LockAction=0 | Specifies whether, on failure to get a lock, the server does nothing other than report the error (default setting of 0), or drops all locks on tables in this query (set to 1) or drops all outstanding locks (a setting of 2). |
| LockTimeout=-1 | Determines the number of seconds that U/SQL will wait for a locked record to become available. This is done by sleeping for one second between attempts and then retrying. If set to -1 (the default) U/SQL will wait forever. If set to 0, then any locked record will be |

| | |
|-------------------|---|
| | reported instantly. |
| MaxPlans=40000 | Specifies how many attempts the plan optimiser will make before giving up. 40000 is every combination of 8 tables, and beyond that the time taken begins to become unpleasant. If this number of plans have been taken, the likely cause is that the user is testing a series of tables, all identical, so an arbitrary choice is as good as anything. If any real-life situation occurs where more plans are needed, simply increase the number. |
| MaxRows=-1 | Specifies how many rows to return by default from any query. -1 and 0 both represent infinity, so all rows are returned. |
| NoDuplicateWarn=N | Don't give a warning on duplicate inserts. |
| NullDates | In Sculptor or BB-ISAM if a date is 0, set it to NULL if this directive is set to Y . |
| NullUnknown | If set to BB-ISAM , NULL values operate as per the SQL standard, that is, a NULL value counts as unknown and neither matches, nor does not match, anything. For example, A=3 and A<>3 will both fail if A is NULL. Only "A is NULL" will be TRUE. |
| NUM2CHAR | Set to L to left justify string numeric values. |
| ODBCDLLLib | Specifies what directory to add to the SHLIB_PATH or LD_LIBRARY_PATH to locate the driver manager (and any subsequent libraries). |
| ODBCManager | Specifies what name to use for the ODBC driver manager. |
| PartialIndex=N | If set to Y , the query plan can then include a second key section from further along the key. For example, the query " select * from salehist where year=94 and custcode='C001' " would normally use year alone to perform the selection. However, if there are many customers and only 12 periods (as is the case) it makes more sense to perform jumps between 94-1-C001 and 94-2- |

| | |
|--------------------------|---|
| | C002, and so on, so 12 indexed starts are made to locations where the potential result sets will be found. |
| Qualifier="" | If set all table owners will be produced by concatenating qualifier and owner with the contents of this string. So if qualifier is set to # then a table q1.a1.customer can be referenced as q1#a1.customer , and will appear in SQLTables() results as being qualifier=NULL, author="q1#a1", tablename="customer" . |
| SculptorDelayIndexOpen=N | Do not open all the indices associated with a file when initialising the file, but wait and see which ones are actually used. When a query is using several files and they have many indices each this is required to stop an error -1 being returned during the query initialisation stage caused by running out of the maximum 32 'units' available within Sculptor, one of which is used for each open file or index. |
| SetInIndexUse=N | The use of indices when comparing to a subquery has been fixed post 300, but this option, when changed to Y , enables a further level of index use. With the following query: 'select custcode from customer where custcode in (select custcode from customer where custcode like "C0%1")' may be stupid, but is indicative of the speed improvement. For C001, the IN clause is satisfied. For C002, it is not. The next available value is C011, so the outer query jumps to C011. When C012 is read, there are no more available values, so the outer query terminates with no more records. Thus 4 records have been read rather than all records in the customer file scanned. |
| SQL_OUTER_JOINS=F | Specifies what value is returned when an ODBC GetInfo call asks about outer join capabilities. This has been made configurable because not all utilities accept F , requiring Y instead to enable outer joins. Microsoft Excel requires Y . |
| TxnIsolationLevel=1 | If set to 1 (SQL_TXN_READ_UNCOMMITTED), dirty reads may be carried out, which means that non-committed rows may be read freely. If set to 2 (SQL_TXN_READ_COMMITTED), rows which have outstanding locks on them will not be readable, and will block the statement as described under the LockTimeout |

| | |
|-----------------|---|
| | parameter. This is only available within Micro Focus. |
| WeekCalculation | <p>Set to U, W, V or ODBC. This determines how week numbers are calculated. In basic terms:</p> <p>U - starts at week 0 for the first week or part-week of the year, with the week number changing to the next week on the Sunday.</p> <p>W - starts at 0 and changes week number on a Monday.</p> <p>V - differs from U only in that the effective week 0 of this year is amended to be week 53 of the previous year.</p> <p>ODBC - is exactly like U, but starting from week 1 rather than from week 0. This is the default setting.</p> |

Validation Rules for Security Directives

Note: The validation of security directives applies only on UNIX.

The directives, **Security** and **SecurityFile** are checked during the validation procedure of the **usqlsd.ini** file.

Validation Rules

- The values of the **Security** and **SecurityFile** directives for a specific DSD are based on the following two rules:
 - If the directives are defined in the **Data Source Defaults** section but not in the specific DSD section then the values in the **Data Source Defaults** section are used. For example:

```
[Data Source Defaults]
Security=None

[books.udd]
Directory=../example
```

When accessing **books.udd** the Security setting (**Security=None**) in the **Data Source Defaults** section is applied, so security is not used for this data source.

- If the directives are defined in both the Data Source Defaults section and the specific DSD section, then the settings in the DSD section take precedence and are used.

```
[Data Source Defaults]
Security=None

[books.udd]
Directory=../example
Security=Host
```

When accessing **books.udd** the Security setting (**Security=Host**) in the **books.udd** section takes precedence, so Operating system level security is used.

- The validation procedure validates the **Configuration** Section, the **Data Source Defaults** Section and the DSD Section of the current DSD that is going to be accessed. Any wrong definition in any other section is ignored. For example:

```
[Configuration Settings]
DefaultServer=sgpower1
DefaultPort=7002
LogLevel=0
License=/u/dev/banner/usqlcs/general.lic
[Data Source Defaults]
[books.udd]
Directory=../example
```

```
Security=None  
[books32.udd]  
Dictionary=../example/books.udd  
Dirtory=../example
```

If you are accessing **books.udd** then the error in the **books32.udd** section (Dirtory is a misspelling) is ignored.

- The directive **Security** can be set either to 'Host' or 'None' without setting any other directive. See example above.
- Whenever the **SecurityFile** directive is defined, the directive **Security** must be set to 'Host'. See example below.
- If the **SecurityFile** directive is set to '*' for a specific DSD then a User Access section must be defined for that specific DSD in the **usqlsd.ini** file. For example:

```
[books.udd]  
Directory=../example  
Security=Host  
SecurityFile=*  
[books.udd_Access]  
jonsmsth=full
```

COGNOS Impromptu Outer Join Directives

If you are experiencing problems with nested outer joins, the following changes must be applied to the Impromptu **cogdmod.ini** file.

Changes needed to support the Transoft ODBC driver

```
[Exceptions Joins DRIVER-TSODBC32.DLL]
Left_Outer=T
nested_Outer=T
One_Outer=F
Optnl_Tbl_restrict=F
[Exceptions Joins DRIVER-TSODBC.DLL]
Left_Outer=T
nested_Outer=T
One_Outer=F
Optnl_Tbl_restrict=F
```

These entries were supplied by Cognos Ltd.

The USQLCS.LOG File and Error Reporting

The USQLCS.LOG File and Log Levels

U/SQL Single and Multiple-tier logs various levels of messages to the log file **USQLCS.LOG**. New information is always appended to an existing **USQLCS.LOG** file.

The log file can be queried using the U/SQL Client based Interactive U/SQL utility, [Win U/SQLi](#). Refer to the section [How to Query the Log File](#).

Single-tier

On Single-tier the log file directory and the log level are controlled from the [U/SQL Administrator](#) using the **Advanced Options** setting. **LogFileDir** and **LogLevel** both appear on installation as blank, but default to **C:\temp** and **0** respectively.

Multiple-tier on Windows

The log file directory and the log level are set using the [U/SQL Service Manager](#). They appear within the **Configuration Settings** section, being common across all data sources. **LogFileDir** defaults to the installation directory of U/SQL, while **LogLevel** defaults to 0.

Multi-tier on UNIX

The log file directory and the log level are set within the **usqlsd.ini** file, either using the **serv_setup.sh** script or simply by editing the file directly. The default settings are:

```
[Configuration Settings]
LogFileDir=/usr/usqls/bin
LogLevel=0
```

Note: If you change the entries for **LogFileDir** or **LogLevel** then ensure you stop and re-start the U/SQL Server for these changes to take effect.

Log Levels

There are five logging levels (0-4). For normal usage, it is recommended that only levels 0 and 1 are used. Do not use higher levels unless explicitly requested to do so, as these are meant for troubleshooting and debugging. Even at level 2, the log file will grow very rapidly, and level 4 records at the network messaging level and can seriously affect performance. The default log level is 0. The following table describes the types of message logging available:

| LogLevel | Identifier | Type | Example |
|----------|------------|------------|---|
| 0 | ERR | Error | Cannot open file: CUSTOMER |
| 1 | WRN | Warning | Field length of column NAME exceeds maximum of 30 |
| 2 | CON | Connection | Connect: DSN=books.udd;UID=Admin |

| | | | |
|---|-----|-------------|--|
| 3 | INF | Information | SQLExecDirect: SELECT CODE FROM CUSTOMER |
| 4 | DBG | Debug | [CUSTOMER] -1 002 - isopen |

Note: With **LogLevel 4** you obtain all the error messages described in the above section.

For further details on what messages are captured in the log file for each log level, how you interpret them and how you access the log file, refer to the section [How to Query the Log File](#).

Log File Messages

Messages are logged according to their message type and the current setting of **LogLevel**. For example:

- If LogLevel has been set to **2**, messages of type **CON**, **WRN** and **ERR** are logged.
- If LogLevel has been set to **4**, messages of all types are logged.

The format of a logged message is as follows:

```
msg_type: dd/mmm/yy hh:mm:ss login_name pid[ src_fname src_lineno]:
[--- errno: errno]
--- msg_txt
```

where:

| | |
|-------------------|---|
| <i>msg_type</i> | Message type identifier (ERR , WRN , CON , INF , DBG). |
| <i>dd/mmm/yy</i> | Date with an alphabetic month. |
| <i>hh:mm:ss</i> | Time in 24-hour format. |
| <i>login_name</i> | Login username. |
| <i>pid</i> | Process-id. |
| <i>src_fname</i> | Source file name (when msg_type is DBG or LogLevel is 4). |
| <i>src_lineno</i> | Line number within source file (when msg_type is DBG or LogLevel is 4). |
| <i>errno</i> | 'C' language error number. Note that in many cases this will not be relevant - it is included for the few cases where it is relevant. |
| <i>msg_txt</i> | Message text. |

For example:

Transoft U/SQL User Guide

```
DBG: 10/Sep/97 16:29:56 kernighan 14453 cisam.c 572:
--- [CUSTOMER ] 00 100 - iswrite      [C016\x00\x00\x00\x00\x00\x00
 ]
ERR: 10/Sep/97 16:29:56 kernighan 14453 procmsg1.c 271:
--- errno: 0.
--- [C-ISAM] (log: 14453-162956) 100: Duplicate key condition
encountered.
DBG: 10/Sep/97 16:29:56 kernighan 14453 objutil.c 74:
--- PostError: State=S1000 Errnum=413 Errmsg=[C-ISAM] (log: 14453-
162956) 100: Duplicate key condition encountered.
ERR: 10/Sep/97 16:29:56 kernighan 14453 server.c 59:
--- errno: 0.
--- SQLError: S1000 (413) [Transoft][TSODBC][usqlsd][C-ISAM] (log:
14453-162956) 100: Duplicate key condition encountered.
DBG: 10/Sep/97 16:29:56 kernighan 14453 svrmain.c 199:
--- Retn: -1 = SQL_ERROR
```

The message text (*msg_txt*) of messages of type **DBG** tend to be of a fixed format when originating from within data source drivers. Due to individual data source characteristics, detailed debug level message logging varies from one data source to the next.

ACUCOBOL Log File Messages

For the ACUCOBOL data source driver, *msg_txt* takes on the following form:

```
--- [data_fname] file_ptr f_errno f_int_errno - fnc_name [rec_buf]
```

where:

| | |
|--------------------|--|
| <i>data_fname</i> | Data file name. |
| <i>file_ptr</i> | ACUCOBOL file pointer in hexadecimal. |
| <i>f_errno</i> | ACUCOBOL error number of last function called. |
| <i>f_int_errno</i> | ACUCOBOL file system specific error number when <i>f_errno</i> is set to E_INTERFACE . |
| <i>fnc_name</i> | ACUCOBOL file system interface function name. |
| <i>rec_buf</i> | Contents of data file record buffer. Unprintable characters are in standard hexadecimal string format: <code>\xhh</code> , where <i>hh</i> is one or more hexadecimal digits (0...9, a...f). |

For example:

```
DBG: 09/Jul/97 10:35:47 kernighan 162956 acuerror.c 149:
--- [STOCK ] 0x51b4b4 00 00 - i_next      [005269Concise Guide
to MS-DOS ].
```

Micro Focus COBOL Log File Messages

For the Micro Focus COBOL data source driver, *msg_txt* takes on the following form:

```
--- [data_fname] fcd_ptr file_status - op_code [rec_buf]
```

where:

| | |
|--------------------|--|
| <i>data_fname</i> | Data file name. |
| <i>fcd_ptr</i> | Micro Focus COBOL FCD (File Control Description) pointer in hexadecimal. |
| <i>file_status</i> | Micro Focus COBOL file status code. |
| <i>op_code</i> | Micro Focus COBOL file handler operation code. |
| <i>rec_buf</i> | Contents of data file record buffer. Unprintable characters are in standard hexadecimal string format: <code>\xhh</code> , where <i>hh</i> is one or more hexadecimal digits (0...9, a...f). |

For example:

```
DBG: 09/Jul/97 10:27:55 kernighan 162956 mffile.c 823:
--- [STOCK      ] 0x568968 0/0  - MF_READ_SEQ_NO_LOCK
   [005269Concise Guide t].
```

C-ISAM Log File Messages

For the C-ISAM data source driver, *msg_txt* takes on the following form:

```
--- [data_fname] isfd iserrno - fnc_name [rec_buf]
```

where:

| | |
|-------------------|--|
| <i>data_fname</i> | Data file name. |
| <i>isfd</i> | C-ISAM file descriptor. |
| <i>iserrno</i> | C-ISAM error number. |
| <i>fnc_name</i> | C-ISAM API function name. |
| <i>rec_buf</i> | Contents of data file record buffer. Unprintable characters are in standard hexadecimal string format: <code>\xhh</code> , where <i>hh</i> is one or more hexadecimal digits (0...9, a...f). |

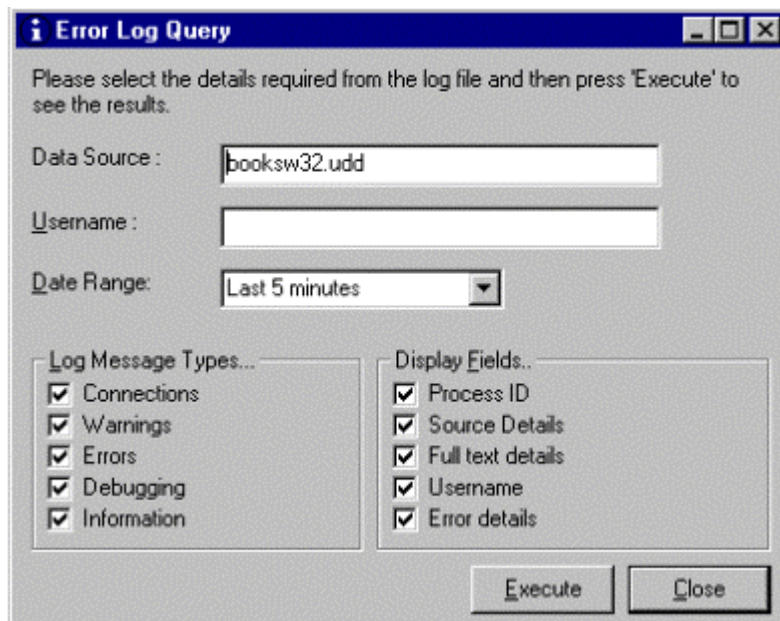
For example:

```
DBG: 09/Jul/97 10:40:29 kernighan 162956 cisam.c 557:  
--- [STOCK      ] 00 000 - isread      [005269Concise Guide to MS-DOS  
].
```

How to Query the Log File

The **usqlcs.log** log file can be queried using the U/SQL Client based Interactive U/SQL utility, **Win U/SQLi**. Select this program from the program group.

Select the **Log** command from the **View** menu . The **Error Log Query** dialog box is displayed:



Select the log information you want to obtain:

- For a particular Data Source
- For a particular Username
- For a Date Range
- For various Log Message Types (equivalent to the LogLevels)
- With various Display Fields

Click **Execute** and Win U/SQLi generates the necessary SQL query to select the information you have requested and automatically executes the query displaying the results in the **Results** window.

The contents of the messages displayed are determined by the data source and the level of message type selected: refer to the above section.

Error Messages

This section covers the following topics:

- [ODBC Error Handling - A Brief Description](#)
- [U/SQL Error Messages](#)
- [The Error Message File - usqlcs.msg](#)
- [Error Reporting Format](#)
- [SQL Error Messages.](#)

ODBC Error Handling - A Brief Description

ODBC error handling is accomplished by one or more calls to **SQLERROR()** when a previous ODBC function call returns **SQL_ERROR** to the ODBC-enabled client application. The client application requests one of three types of error depending on the type of the previous ODBC function that has failed. It does this by passing either a NULL or a valid handle for either the environment, connection or statement handle arguments of **SQLERROR()** (see *Microsoft ODBC 2.0 Programmer's Reference* and *SDK Guide* for details). **SQLERROR()** then returns an error message from the top of either the environment, connection or statement error information stacks. The application continues to call **SQLERROR()**, retrieving successive levels of error information, whilst its return code is **SQL_SUCCESS**, and stops when the return code changes to **SQL_NO_DATA_FOUND**.

Error message text returned by **SQLERROR()** can be in one of two formats:

- For errors that do not occur in a data source:
`[vendor_identifier][ODBC_component_identifier](log: log_number)
component_supplied_text`
- For errors that occur in a data source:
`[vendor_identifier][ODBC_component_identifier][data_source_identifier]
(log: log_number)data_source_supplied_text`

U/SQL Error Messages

U/SQL reports a range of error messages:

- Some are produced directly by the U/SQL Engine. These can be terse, for example:

```
[Transoft][TSODBC] Error State 01004
```

The SQLSTATE number 01004 means Data truncated, see the section SQL Error Messages at the end of this chapter for details of these SQLSTATE codes and messages.

- Most of the error messages you are likely to encounter are contained in the message file **usqlcs.msg**. These are detailed below in the section [The Error Message File - usqlcs.msg](#).
- A log level directive, **LogLevel**, can be set that produces various levels of messages to the log file, **usqlcs.log**, that can assist in debugging particular situations. Refer to the [The USQLCS.LOG File and Log Levels](#)

section.

The Error Message File - usqlcs.msg

Certain error messages are contained in the message file **usqlcs.msg** and cover the following sections:

- Standard messages, for example:
 Cannot open file: <file_name>
- Dictionary maintenance messages
- Transaction processing messages
- GRANT and REVOKE security messages
- Data source driver messages
- Data source driver messages with file handler error codes.

The last message category is used immediately after file input/output has taken place, and the message itself is preceded by the file handler (for example, C-ISAM, Micro Focus COBOL, PRO-ISAM, etc.) error code. Consequently all data source drivers using **usqlcs.msg**, include the file handler error code. For example:

| | |
|-------------------|---|
| Informix C-ISAM | 100: Duplicate key condition encountered. |
| Micro Focus COBOL | 2/2: Duplicate key condition encountered. |
| PRO-ISAM | 108: Duplicate key condition encountered. |

The **usqlcs.msg** file can be viewed for a complete list of the messages. For Single-tier, it is found in the **WINDOWS\SYSTEM** directory and, for Multiple-tier for UNIX and Windows NT Server, it is found in the **bin** directory below the U/SQL Server software installation base directory.

Error Reporting Format

Error message text can be in one of two formats. For errors that do not occur in a data source:

[vendor_identifier][ODBC_component_identifier](log: log_number)
component_supplied_text

For errors that occur in a data source:

[vendor_identifier][ODBC_component_identifier][data_source_identifier] (log:
log_number) data_source_supplied_text

where (*log_number*) is of the form - (*pid-hhmmss*)

where:

pid Process-id.

hhmmss Time in 24-hour format.

Example error messages are:

- **Multiple-tier. For example:**

```
[Transoft][TSODBC][usqlsd][C-ISAM](log: 14453-162956) 100: Duplicate key condition encountered.
```
- **Single-tier. For example:**

```
[Transoft][TSENG32][C-ISAM](log: 14453-162956) 100: Duplicate key condition encountered.
```
- **In the UNIX based Interactive U/SQL utility, [usqli](#). For example:**

```
*** Error: (log: 14453-162956) 100: Duplicate key condition encountered.
```
- **In the U/SQL Client based Interactive U/SQL utility, [Win U/SQLi](#). For example:**

```
*** Error: (log: 14453-162956) 100: Duplicate key condition encountered.
```
- **Note that the file-handler name and error code only appear as a result of a failure within the file-handler. Examples from other file-handlers are:**

```
[Transoft][TSODBC][usqlsd][MF](log: 14453-162956) 2/2: Duplicate key condition encountered.
```

```
[Transoft][TSODBC][usqlsd][PRO-ISAM](log: 14453-162956) 108: Duplicate key condition encountered.
```
- **Non file-handler specific errors appear without the file-handler name; for example:**

```
[Transoft][TSODBC][usqlsd](log: 14453-162956) Invalid record expression in table CUSTOMER.
```
- **The following is an example of an error message displayed by Microsoft Access using Multiple-tier U/SQL C-ISAM:**



SQL Error Messages

U/SQL outputs ODBC SQLERROR that returns SQLSTATE values.

The character string value returned for an SQLSTATE consists of a two-character class value followed by a three-character subclass value. A class value of '01' indicates a warning and is accompanied by a return code of SQL_SUCCESS_WITH_INFO. Class values other than '01', except for the class 'IM', indicate an error and are accompanied by a return code of SQL_ERROR. The class 'IM' is specific to warnings and errors that derive from the implementation of ODBC itself. The subclass value '000' in any class is for implementation defined conditions within the given class. The assignment of class and subclass values is defined by ANSI SQL-92.

Where appropriate the SQLSTATE values will be return in error messages from the U/SQL Engine with the **LogLevel of 0**, the default.

Example SQLSTATE values, error messages and the ODBC function calls that may have returned them are shown below:

| SQLSTATE | Error message | Can be returned from |
|-----------------|-------------------------------------|--|
| 01000 | General warning | All ODBC functions except: SQLAllocEnv, SQLError |
| 01002 | Disconnect error | SQLDisconnect |
| 01S00 | Invalid connection string attribute | SQLBrowseConnect SQLDriverConnect |
| 01S01 | Error in row | SQLExtendedFetch SQLSetPos |

SQL Support

SQL Syntax Supported

U/SQL Revision 3.00 conforms to an ODBC Version 2 driver and works with the ODBC Driver Manager Version 3. It supports ANSI '89 and ANSI '92 Compliant SQL syntax with the ODBC shorthand SQL syntax for scalar functions, dates, outer-joins and so on, which use the '{' (brace) syntax.

Although it is not possible to extend the underlying legacy data by making use of the CREATE TABLE syntax, it is possible to use this syntax to create tables and indexes in the UDD itself to hold temporary or other information. Refer to the [Hints and Tips](#) section.

U/SQL supports, for example:

- **SELECT... WHERE...** statements with **AND, OR =, !=, >, <, >=, <=, BETWEEN, NOT, LIKE**
- Joining tables with **SELECT** (including Cartesian product)
- Arithmetic functions in **SELECT** statements (for example, SELECT (column_name + 1) * 2, and so on)
- String functions in **SELECT** statements
- **GROUP BY... HAVING...** clauses in **SELECT** statements
- Set functions (**AVG, SUM, MAX, MIN, COUNT**)
- **ORDER BY** clauses
- **SELECT DISTINCT**
- Nested queries (**SELECT** within **SELECT**)
- Date arithmetic.

U/SQL also supports a wide means of manipulating data. But you must not have Read-Only set, either via your license or via the **ReadOnly** directive. Key data manipulation statements are:

- **INSERT** statements
- **UPDATE** statements
- **DELETE** statements.

For further information on SQL syntax and facilities supported by a Version 2 ODBC driver, refer to the *Microsoft ODBC 2.0 Programmer's Reference* and *SDK Guide* for details.

General

- The syntax:

```
"select top <n> ...<query> "
```

is supported, so

```
"select top 3 * from customer"
```

will only return the first three rows of the customer file. This can be useful when developing queries. This functionality is also supported as a directive **MaxRows=n**, where n is the number of rows to be returned (at most).

Sample Queries

Sample SQL Syntax for Querying and Manipulating Data

The following sections show examples of SQL syntax for querying and manipulating data for ODBC 2 compliance.

The examples shown make use of the Books Demonstration data provided with your data source. In general, you will be able to reproduce these examples yourself.

SELECT

| Example | SQL Statement(s) | Description |
|----------|---|--------------------------|
| 89SEL090 | <pre>select custcode as Code, cust_name as Name, cust_region as Region from customer;</pre> | Correlated column names. |

JOINS

| Example | SQL Statement(s) | Description |
|----------|---|--|
| 89JOI100 | <pre>select cust_name, odate, stknum from customer, orders, oline where custcode = custcode and orders.ordno = oline.ordno order by cust_name, odate, stknum;</pre> | 3-way natural join (ordered on all fields) |

PREDICATES

| Example | SQL Statement(s) | Description |
|----------|---|--|
| 89PRE034 | <pre>select name from customer where name between "A" and "N" and code between "C001" and "C005";</pre> | BETWEEN with AND on two different fields |
| 89PRE040 | <pre>select * from customer where cust_region in ('EAST');</pre> | Simple IN (single value) |
| 89PRE057 | <pre>select * from customer where not (cust_name like 'The %');</pre> | NOT (LIKE %) |
| 89PRE073 | <pre>select * from customer where NOT (cust_region is</pre> | NOT (IS NOT NULL) |

| | | |
|----------|--|-------------------------------------|
| | NOT NULL); | |
| 89CND040 | select * from stock where not (price < 20.45 or price > 30); | NOT (< or >) (inclusive between) |

FUNCTIONS

| Example | SQL Statement(s) | Description |
|----------|--|-----------------|
| 89FUN010 | select count (distinct cust_region) from customer; | COUNT(DISTINCT) |
| 89FUN020 | select avg(target) from budget; | AVG |
| 89FUN040 | select max(target) from budget; | MAX |
| 89FUN060 | select min(target) from budget; | MIN |
| 89FUN080 | select sum(price), sum(qty) from stock; | SUM |

Rounding

| Example | SQL Statement(s) | Description |
|----------|--|------------------------------|
| 89RND010 | create table fred (code numeric (9,4)); insert into fred values (3); select sum(-code) from fred; | Sum rounding of single value |
| 89RND020 | select 5000.04 + 5001.04 + 5000.04 + -15000.12 from customer; | Multiple value addition |

ORDER BY

| Example | SQL Statement(s) | Description |
|----------|---|---|
| 89ORD020 | select a.cust_region, a.cust_name, b.custcode, b.odate from customer a, orders b where b.custcode=a.custcode order by a.cust_region, a.cust_name, b.custcode, b.odate desc; | Order on columns produced by a join of two tables, with one of the ordered columns being specified as descending. |

| | | |
|----------|--|---|
| 89ORD053 | select distinct custcode from salehist order by custcode desc; | select distinct instances of 2nd alternate key and reverse order |
|----------|--|---|

GROUP BY... HAVING...

| Example | SQL Statement(s) | Description |
|----------|--|--|
| 89GRP071 | select cust_name, cust_region, category from customer, budget where region = cust_region group by cust_name, cust_region, category having sum(target) > 8000 and sum(traget) < 12000; | GROUP BY... HAVING... with AND predicates (across two tables). |

ERROR MESSAGES

These are a few sample error messages:

| Example | SQL Statement(s) | Description |
|----------|--|---|
| 89ERR000 | select * from no_table; | ERROR : Table does not exist |
| 89ERR010 | select svalue, year, no_column from salehist; | ERROR : Column does not exist in table |
| 89ERR020 | Select target, category, region; | ERROR : No from clause |
| 89ERR030 | select from customer; | ERROR : No fields in select |
| 89ERR040 | select category, avg(target) from budget; | ERROR : Column not grouped |
| 89ERR050 | select sum(category) from budget; | ERROR : Invalid column spec. |

SUB-QUERIES

| Example | SQL Statement(s) | Description |
|----------|---|------------------|
| 89GRP071 | select a.stknum, a.stock_title, a.qty from stock a where a.stknum not in (select b.stknum from oline b where b.qty > 1); | WHERE ...NOT IN. |

INSERTS

| Example | SQL Statement(s) | Description |
|---------|------------------|-------------|
|---------|------------------|-------------|

| | | |
|----------|---|---|
| 89INS013 | <pre>insert into oline (ordno, lineno, stknum, qty) select ordno, lineno+1, stknum, qty from oline where ordno=123001 and lineno=3;</pre> | <p>INSERT..SELECT FROM WHERE. Insert one row based upon the values returned by the subquery</p> |
|----------|---|---|

UPDATES

Statements in italics are select statements for viewing 'before' and 'after' the update to see if it has worked correctly.

| Example | SQL Statement(s) | Description |
|----------|---|---|
| 89UPD014 | <pre><i>select stock.stknum, stock.price from stock, salehist, customer where cust_region="SOUTH" and salehist.custcode=customer.custcode and stock.stknum=salehist.stknum;</i> update stock A set A.price = 700 where A.stknum in (select distinct B.stknum from salehist B, customer C where C.cust_region="SOUTH" and B.custcode=C.custcode); (Same post-select as pre-select)</pre> | <p>UPDATE WHERE IN. Update column to fixed value contained in update statement, where the column matches the subquery specification. 'Pre-Select' & 'Post-Select' return the values that would be found by the subquery, which should all show the same value after the update.</p> |

DELETES

Statements in italics are select statements for viewing 'before' and 'after' the update to see if it has worked correctly.

| Example | SQL Statement(s) | Description |
|----------|---|--|
| 89DEL021 | <pre><i>select distinct customer.custcode, customer.cust_name from customer, orders where customer.custcode not in(select customer.custcode from customer, orders where customer.custcode=orders.custcode);</i> delete from customer where customer.custcode not in(select customer.custcode from customer, orders where customer.custcode=orders.custcode); (Same post-select as pre-select)</pre> | <p>DELETE WHERE NOT IN. Delete all rows NOT matched in subquery.</p> |

UNIONS

| Example | SQL Statement(s) | Description |
|----------|--|--|
| 89UNI020 | <pre>select * from customer where cust_region = 'EAST' UNION ALL select * from customer where cust_name >'B';</pre> | UNION ALL test. Returns all rows in customer table, with duplicates. |

OUTER JOINS

To obtain the full range of support for Outer Joins, please refer to the Release Notice, supplied with the U/SQL Client software.

| Example | SQL Statement(s) | Description |
|----------|---|----------------------------------|
| 92JOI000 | <pre>select cust_name, customer.cust_region, orders.odate from customer left outer join orders on customer.custcode = orders.custcode;</pre> | Full ANSI'89 syntax |
| 92JOI001 | <pre>select cust_name, customer.cust_region, orders.odate from customer left join orders on customer.custcode = orders.custcode;</pre> | Short ANSI'89 syntax |
| 92JOI002 | <pre>select cust_name, customer.cust_region, orders.odate from -- (*vendor(microsoft),product(odbc) oj customer left outer join orders on customer.custcode = orders.custcode*);</pre> | Full ODBC syntax |
| 92JOI003 | <pre>select cust_name, customer.cust_region, orders.odate from {oj customer left outer join orders on customer.custcode = orders.custcode};</pre> | Short ODBC syntax |
| 92JOI010 | <pre>insert into orders values (123029,"C007","1992-12-11") ; insert into orders values (123030,"C015","1992-12-11"); select * from orders left outer join oline on orders.ordno = oline.ordno;</pre> | SELECT * with numeric outer join |
| 92JOI011 | <pre>insert into orders values (123029,"C007","1992-12-11"); insert into orders values</pre> | outer join functions |

| | | |
|----------|---|--------------------------------|
| | <pre>(123030,"C015","1992-12-11"); select count(*), min(qty), max(qty), avg(lineno), sum(qty) from orders left outer join oline on orders.ordno = oline.ordno;</pre> | |
| 92JOI020 | <pre>insert into orders values (123029,"C007","1992-12-11"); insert into orders values (123030,"C015","1992-12-11"); select * from orders left outer join oline using ordno;</pre> | USING syntax |
| 92JOI030 | <pre>select cust_name, customer.cust_region, orders.odate from customer left outer join orders on customer.custcode = orders.custcode where cust_name > "n";</pre> | Simple predicate in outer join |
| 92JOI040 | <pre>create table sales_table (salecode char(4), goods char(4), value integer); create unique index SALECODEK0 on sales_table(salecode); insert into sales_table values ('C002', 'G001', 123); insert into sales_table values ('C002', 'G002', null); insert into sales_table values ('C002', 'G003', 345); select cust_name, custcode, salecode, goods, value from {oj customer left outer join sales_table on custcode = salecode }; drop table sales_table;</pre> | |

Scalar String Functions

Scalar functions allow scaling conversion to be carried out on data being returned.

The scalar functions are specified in the *Microsoft ODBC Programmer's Reference* and *SDK Guide* to which reference should be made as there are too many to list in a document such as this. However, some examples are:

| Example | SQL Statement(s) | Description |
|----------|--|----------------------------|
| 92STR000 | <pre>select cust_name, {fn ascii("A")}, {fn ASCII(CUST_NAME)}, {fn</pre> | ODBC ASCII and CHAR scalar |

| | | |
|----------|--|---|
| | CHAR(65)), {fn char({fn ascii(cust_name)})} from customer; | functions. |
| 92STR010 | select cust_name, {fn lcase("ABCdEF")}, {fn ucase("abcDef")}, {fn lcase(cust_name)}, {fn ucase(cust_name)} from customer; | ODBC LCASE and UCASE scalar functions. |
| 92STR020 | select cust_name, custcode, {fn left(cust_name,10)}, {fn eft("ABCDEFGHI",5)}, {fn right(custcode,3)}, {fn right("ABCDEFGHI",5)}, {{fn right({fn left(cust_name,13)}, 5)} from customer; | ODBC LEFT and RIGHT scalar functions |
| 92STR030 | select cust_name, {fn length("ABcdEF h")}, {fn length(cust_name)}, {fn rtrim(cust_name)}, {fn length({fn rtrim(cust_name)})} from customer; | ODBC LENGTH and RTRIM scalar functions |
| 92STR040 | select cust_name, {fn soundex(cust_name)} from customer; | SOUNDEX function |
| 92STR041 | select {fn soundex("Smith")}, {fn soundex("Smyth")}, {fn SOUNDEX("Smiff")} from customer where custcode="C001"; | SOUNDEX function |
| 92STR050 | select cust_name, {fn replace(cust_name,"Book",'Celery')} | REPLACE function |
| 92STR060 | select cust_name, {fn locate('Book',cust_name)} from customer; | LOCATE function |
| 92STR070 | select {fn concat(category, region)} from orders; | CONCAT function |
| 92STR080 | select cust_name, {fn substring("ABCdEFghiJKLMN", 4, 5)}, {fn substring(cust_name, 10, 6)} from customer; | SUBSTRING function |
| 92NUM010 | select svalue, {fn pi()}, {fn radians(svalue)}, svalue * {fn pi()} / 180 from salehist where svalue<=180; | PI and RADIANS function NB. x*Pi/180 converts degrees to radians |
| 92NUM020 | select svalue, {fn sin({fn radians(svalue)})}, {fn cos(svalue * {fn pi()} / 180)}, {fn tan({fn radians(svalue)})} from | SIN, COS, and TAN functions (with RADIANS function) |

| | | |
|----------|---|---|
| | salehist where svalue<=180; | |
| 92DAT010 | select {fn curdate()} from customer where custcode="C002"; | CURDATE function. NB. Test uses UNIX 'date' function to check result |
| 92DAT020 | select odate, {fn dayname(odate)}, {fn dayofmonth(odate)}, {fn dayofweek(odate)}, {fn dayofyear(odate)} from orders; | DAYNAME, DAYOFMONTH, DAYOFMONTH, and DAYOFYEAR functions |
| 92SYS010 | select {fn database()} from customer; | DATABASE function |
| 92SYS020 | select {fn ifnull(region,"None")} from customer; | IFNULL function |
| 92CNV010 | select {fn convert(odate,SQL_TIMESTAMP)} from orders; | CONVERT from SQL_DATE to SQL_TIMESTAMP |
| 92CNV020 | select (fn convert({fn now()} , SQL_DATE)} from customer where custcode="C002"; | NOW() function and CONVERT to SQL_DATE from SQL_TIMESTAMP |

READ_ONLY Views

The following syntax is now supported:

```
create view <viewname> [ ( columns-list... ) ] as <select statement>
and
drop view <viewname>
```

The create view statement generates a read-only view which may be used as a table in its own right. For example:

```
create view eastcustomers as select * from customer where
cust_region="EAST";
select * from eastcustomers;
```

returns:

| CUSTCODE | CUST_NAME | CUST_REGION |
|----------|--------------------------|-------------|
| ----- | ----- | ----- |
| C001 | Alden & Blackwell | EAST |
| C005 | Dillons the Bookstore | EAST |
| C014 | Volume 1 bookshops | EAST |

3 records retrieved

Alternatively:

```
create view eastcustomer2 ( code, name ) as select custcode,
cust_name from customer where cust_region="EAST";
select * from eastcustomer2;
```

returns:

| code | name |
|------|-----------------------|
| ---- | ---- |
| C001 | Alden & Blackwell |
| C005 | Dillons the Bookstore |
| C014 | Volume 1 Bookshops |

3 records retrieved

It is also possible to use the **info** command in [usqli](#) to examine view definitions:

```
info eastcustomers
```

returns:

```
U/SQL> info eastcustomers
create view EASTCUSTOMERS (CUSTCODE, CUST_NAME, CUST_REGION) as
select * from customer where cust_region="EAST";
```

```
/* Unique row identifier: CUSTCODE */
```

Note: *The "Unique row identifier" line is used as the primary key if the view is examined via a third-party product such as Microsoft Access.*

Limitations

- **Locking with SELECT FOR UPDATE statement**

U/SQL provides explicit locking at the record level immediately prior to a record being updated or deleted via **UPDATE** and **DELETE** statements. However, the **SELECT FOR UPDATE** statement only locks the current record and not the cursor of records being updated. In order to ensure that all records are locked until the transaction is complete, your own application must make use of a Named Cursor.

For Multiple-tier U/SQL, set the server **LockTimeout=** directive to -1 (the default) to ensure that your application waits indefinitely on any lock set by any other process or ODBC application. See the [Locking](#) section for more information.

- **U/SQL does not allow an aggregate function within another function**

The aggregate function must be the outer-level function. For example, the following is not supported:

```
select {fn ifnull(sum(qty),0)} from oline where lineno=0;
```

This is because the **ifnull()** function is outside the **sum()** function, which is an aggregate function. An aggregate function is any of: **min()**, **max()**, **sum()**, **avg()**, **count()**.

It is permissible to have a function within the aggregate, for example:

```
select sum({fn ifnull(qty,0)}) from oline where lineno=0;
```

Within **GROUP BY** expressions, U/SQL does allow an expression to be grouped, but it must be the exact expression used in the **SELECT** clause. The following is not permitted:

```
select a,b,a*b,sum(c) from myfile group by a,b
```

This is because the expression "a*b" is not grouped, so causes an error to be reported when the query is prepared. It would be necessary to change this to be:

```
select a,b,a*b,sum(c) from myfile group by a,b,a*b
```


Transaction Processing

The ability to perform transaction processing has been implemented in the U/SQL C-ISAM, ACUCOBOL and Micro Focus COBOL data source drivers (DSDs) on UNIX platforms. Informix, the authors of C-ISAM, documents two forms of transaction processing:

- **Non-ANSI**

Transactions are initiated with a begin statement, and terminated by either a commit statement or a rollback statement. Informix C-ISAM conforms to this method of transaction processing.

- **ANSI**

Transactions are implicit. Transactions are explicitly terminated by either a commit statement or a rollback statement. Normally, the first update of data initiates a transaction. Micro Focus COBOL and Microsoft ODBC conform to this method of transaction processing.

The ANSI method of transaction processing has been implemented in U/SQL Adapters.

Activate Transaction Processing

To activate transaction processing, there are additional directives to be placed in UNIX **usqlsd.ini** configuration file.

- `AutoTransProc={Y|N}`

Automatic Transaction Processing. This is set to either **Y** or **N**, and it may be specified within either an individual [**<data source section>**] section or the [Data Source Defaults] section of **usqlsd.ini**. The default is **N**.

AutoTransProc must be set to **Y** for transaction processing to become activated. If it is set to **N** transactions may never be initiated. Once set to **Y** transactions become possible, although it is still necessary either to use "TRANSACTION MODE ON" or the ODBC call `SQLSetStmtOption(...,SQL_AUTOCOMMIT,SQL_AUTOCOMMIT_OFF)` to initiate transactions.

- `TransLogDir=/usr/usqls/bin`

Transaction Log file Directory. This need only be specified for the C-ISAM data source which requires the name of a log file in order to carry out transaction processing. The log filename is **cs_m_trans.log**, and it is located in the directory identified by this entry within the [Configuration Settings] section of **usqlsd.ini**. **cs_m_trans.log** is automatically created if it does not exist; otherwise, it is appended to.

- `FileShare={Y|N}`

For Micro Focus COBOL, transaction processing is only possible if Micro Focus Fileshare is operating. Set the **FileShare** directive to **Y** within either an individual [**<data source section>**] section or the [Data Source Defaults] section of **usqlsd.ini**. The default is **N**.

Note: U/SQL for ACU COBOL supports transaction processing. To enable this you must set the **AutoTransProc** directive.

Transaction Processing Syntax

The transaction processing syntax is as follows:

| | |
|--------------------------------|--|
| TRANSACTION [MODE] {ON OFF} | Commence/Terminate transaction processing. |
| COMMIT [WORK] | Commit a transaction. |
| ROLLBACK [WORK] | Rollback a transaction. |

Both the Interactive U/SQL utilities, the client-based [Win U/SQLi](#) and the UNIX-based [usqli](#), support the transaction processing syntax.

Micro Focus COBOL Data Source Driver

Transaction processing is only possible if Micro Focus Fileshare is operating which is achieved as follows:

1. Log in as root user, and start the process registration daemon:

```
cd /usr/lib/cobol
./ccitcp2 -d
```
2. Log in as a non-root user, and start the **Fileshare server** in the directory containing the data files:

```
fs -s hostname
```
3. Add the following **usqlsd.ini** configuration file entry to either an individual **[<data source section>]** section or the **[Data Source Defaults]** section:

```
FileShare=Y
```
4. Start the U/SQL Server:

```
./start_serv.sh
```

or:

```
./usqlsd port=port_number
```

The SQL Engine instructs the Micro Focus Data Source Driver (DSD) to open and close relevant data files before and after each SQL query respectively. A transaction, in Micro Focus COBOL, commences with the first update of data, and terminates when the file handler is called with an operation code of either commit or rollback. During that time the relevant data files must remain open. The DSD has been modified so that data files remain open during a transaction. However, if a large number of queries are issued involving a large number of data files within any one transaction, it is possible that the maximum number of file descriptors available to any one process may be exceeded.

Rollforward Recovery

It will be necessary to ensure that a C-ISAM transaction log file is backed up as part of any regular data backup procedures. Once this is done, a potentially large log file may be purged. In addition to its use in transaction processing, a log file is used to facilitate a rollforward recovery after a system failure.

Micro Focus Fileshare provides its own form of transaction logging and rollforward recovery. This may be activated at the time Fileshare is started up.

Locking

If the directive **TrueSFU=Y** is set, multiple records may be locked at once in any of C-ISAM, Micro Focus COBOL and ACUCOBOL.

To lock records, a "**SELECT FOR UPDATE**" statement is executed. The behaviour is described below:

- Without **TrueSFU** set (the default): As each row is fetched, it is locked. When the next row is fetched, the first is unlocked and the new one locked. Thus only 1 row can be locked at any one time, and once the query is completed no rows are locked.
- With **TrueSFU** set: As each row is fetched, it is locked as before. However, when the next row is fetched, the first row remains locked. They all remain locked until a **ROLLBACK** or **COMMIT** statement is executed, or (through ODBC) **SQLTransact** is called.

If transaction processing is not in use but **TrueSFU** is set to Y, it is then permissible to use **COMMIT**, **ROLLBACK** or **SQLTransact** to perform a 'dummy transaction end' which unlocks all files associated with the current connection.

The **LockTimeout** INI directive controls how lock U/SQL will wait to acquire a lock before reporting an error. If set to -1 (the default), U/SQL will wait indefinitely. If set to 0, U/SQL will report an error immediately on failing to obtain a lock. If set to any other positive integer, U/SQL will wait for that number of seconds, retrying once per second, until either the time expires and an error is reported or the lock is obtained successfully.

Security

Security

Security is a major issue particularly in the access of multi-user applications. With a variety of ODBC-enabled query tools, using U/SQL, it is important to ensure that only approved users are able to query and modify your application's data.

This section describes:

- [Multiple-tier user connection security](#).
- [Grant and Revoke security](#). The granting and revoking of table and column privileges to users, once they are allowed to connect.
- The role of the Database Administrator (dba).

Multiple-tier Security - User Connection

Multiple-tier Security enforces operating system user connection security. The security features are comprehensive and flexible and cover virtually all user requirements for connection level security.

UNIX and Windows NT Server

Security is controlled at the U/SQL Server through server directive settings either from the **usqlsd.ini** configuration file on UNIX or from the U/SQL Service Manager on Windows. The U/SQL Client automatically detects a secure U/SQL Server and prompts for the user's operating system username and password.

By default, security is off.

Note: The directive names given below appear in the **usqlsd.ini** file on UNIX, but are set from the **Security** tab of the U/SQL Service Manager on Windows Multi-Tier systems. See the [U/SQL Service Manager to set Windows Security Directives](#) section below.

UNIX

When requiring security on UNIX platforms, it is mandatory to start the **usqlsd** U/SQL Server process as root.

Multiple-tier security configuration directives

There are four configuration directives related to Multiple-tier Security, which are set either in the UNIX **usqlsd.ini** configuration file (for further information, refer to the [Configuring Multiple-tier U/SQL](#) section), or using the U/SQL Service Manager on Windows (see the [U/SQL Service Manager to set Windows Security Directives](#) section below).

Security={None|Host}

None: Security is not used for this data source. All further security options are ignored.

Host: Operating system level security is used. The username and password are validated on the host system.

The default setting is Security=None.

BecomeUser={Y|N}

If set to Y, the U/SQL Server changes its user-ID and group-ID to that of the user. This ensures that operating system file permissions take effect. If set to N, the username and password are still validated, but the U/SQL Server does not change user.

Note: To use this option, the U/SQL Server must be started, for UNIX, as root and, for Windows NT Server or Windows 2000, by a user with Administrator privileges.

UnauthorizedAccess={N|Y}

If set to Y, unauthorized connections are allowed, but only have read access. That is any

connection that would otherwise fail due to an invalid username or password, will successfully connect but will have Read Only access. If set to N only validated users have access as normal.

This option may be useful for installations where:

- A. A client-based program has been developed that requires full update access.
- B. There are some "power" users that can be trusted to do updates and where all other users running generic reporting application should only have read access.

If used, this option must be used with **SecurityFile=** option, see below, to specify the user account(s) that have full access. Otherwise, any user knowing a login could get full access.

The default setting is `UnauthorizedAccess=N`.

`SecurityFile=<path_name>` This option is used to limit access to a data source to a specified list of users. You specify a security file, with its path, which contains for each data source a user access section with its list of valid users, see the section [Security File Contents](#) below.

An example security file entry in the UNIX **usqlsd.ini** configuration file might be:

```
SecurityFile=/usr/usqls/security.txt
```

An example security file entry in the Windows NT Server or Windows 2000 Registry file might be (note, the '.SEC' extension):

```
C:\USQLCS\SECURITY.SEC
```

Security file contents

There can be any number of data sources' user access defined in one security file. Each is defined as the Data Source Name (DSN), for example, **books.udd**, with "_Access" appended, in this case **[books.udd_Access]**. Remember the DSN does not have to have a '.udd' extension.

The user access section contains a list of users specifying the access they are permitted in the form:

```
<username>={none|readonly|full}
```

`none` The user has no access.

`readonly` The user can only read from the data files.

`full` The user can read from and write to the data files.

Note: *none, readonly and full must be lowercase.*

The security file must be owned under UNIX by root and under Windows NT Server or Windows 2000 as Administrator and be read only.

UNIX

The UNIX **usqlsd.ini** configuration file can also be used as the security file and setting **SecurityFile=*** defines this. Hence you can specify your security requirements in the one **usqlsd.ini** file or in a separate file, whichever is more convenient.

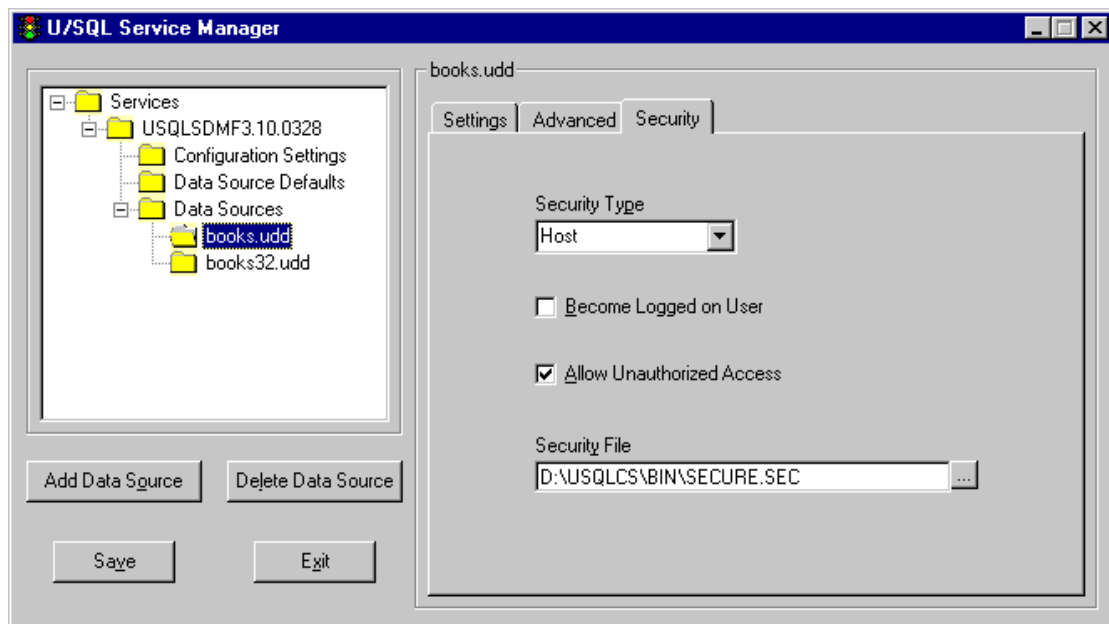
For example, if the Data Source Name is **books.udd**, the user access section in the security file might appear as follows:

```
[books.udd_Access]
root=full
eric=readonly
burt=none
```

If a user does NOT have an entry in the access section or if access is set to none, the user CANNOT connect to the data source. Otherwise the username and password will be validated as usual.

See the **UnauthorizedAccess=** option for an exception to this.

U/SQL Service Manager to set Windows security directives



The above shows example Security settings made via the Windows NT Server U/SQL Service Manager.

Optional lockdown security

Tighter 'lockdown' security is available under both UNIX and Windows NT Server platforms, as follows:

UNIX

You are able to provide tighter security by ensuring only one UNIX **usqlsd.ini** configuration file is used and not other 'local versions'. If the U/SQL Server finds an **usqlsd.ini** file in the precise directory structure **/usr/usqlcs**, then it will look for no other and the following can be implemented to provide this extra level of security:

- A new section **[Data Sources]** must be added, to the **usqlsd.ini** configuration file, which lists the valid data sources. ONLY these can be connected to. For example:

```
[Data Sources]
books=Books demo

[books]
Directory=/usr/usqlcs/example
Dictionary=books.udd
```

- Each data source must specify a **Directory=** option.
- Security is **ON** for all data sources.

To ensure the tightest security, the following is recommended:

1. Create the **/usr/usqlcs** directory and set up in it the **usqlsd.ini** configuration file as specified above.
2. Start the U/SQL Server as **root** or an administrative account that has access to the database files.
3. Set file access permissions so that general users have NO access to the following:
 - The **/usr/usqlcs** directory
 - The **/usr/usqlcs/usqlsd.ini** configuration file
 - **usqli** and **usqlsd** programs
 - The data dictionary '.udd'

4. Set the security configuration parameters as follows:

```
Security=Host
BecomeUser=N
UnauthorizedAccess=N
SecurityFile=* : or a file name
```

Add the list of users to the access section of the security file as specified above.

Windows NT Server and Windows 2000

For Windows NT Server and Windows 2000, lockdown security is implemented by setting the directive **LockdownFile=<filename>** in the [Configuration Settings] section using the U/SQL Service Manager. Refer to the [U/SQL Service Manager](#) section.

The lockdown <filename> is a text file that must contain the section heading **[Data Sources]** followed by a list of valid data source names beneath it. This format is the same as for the **[Data Sources]** section in the **usqlsd.ini** configuration file on UNIX.

GRANT and REVOKE Security

User authorization is required, in general, to prevent some users from accessing:

- Certain tables
- Certain columns
- Via **SELECT**, **INSERT**, **UPDATE** or **DELETE** SQL verbs

In U/SQL revisions 3.00 and above, statements have been provided to support the setting up of users and the granting and revoking of privileges. These statements are entered only via the Interactive U/SQL utilities, [Win U/SQLi](#) or [usqli](#), and are:

- **USER**
- **GRANT**
- **REVOKE**

GRANT and REVOKE security is only invoked when the first USER is defined, who must be the dba (Database Administrator), that is when the following statement is entered via the Win U/SQLi or usqli utilities:

USER dba password

Note: *Ensure you do not lose the dba password as it is not accessible.*

Once the dba user has been created, only he/she can add other users and their passwords. A user once created can change only his/her password (but at any time).

Once User Authorization has been activated, all users will have to enter their UserName and Password when logging on.

Note: *If you are also using Multiple-tier user connection security, then it is advisable to use the same IDs for GRANT and REVOKE user authorization to prevent the need to log in twice.*

The other statements, using the GRANT and REVOKE keywords, are used to define the access privileges. After the dba has set up the UserNames and Passwords, all users have full access rights to all tables. The dba then restricts particular users from certain tables and columns.

This section covers the following topics:

- [User Administration](#)
- [GRANT and REVOKE Syntax](#)
- [Batch Entry](#)
- [Connecting to a Data Source](#)
- [GRANT and REVOKE Security Issues](#)
- [GRANT and REVOKE Error Messages.](#)

User Administration

User IDs and passwords are maintained by using the following two commands:

```
USER dba password
```

```
USER username {password | REMOVE}
```

where password is case-sensitive. These commands can be used via the UNIX and Windows Interactive U/SQL Utilities, [usqli](#) and [Win U/SQLi](#) respectively.

`username` and `password` can be a maximum of 20 characters each.

`dba` is the username for the Database Administrator, and must be in lowercase. GRANT and REVOKE security is activated for the first time within a data source by issuing the first command above. For example:

```
USER dba dbapwd
```

The following takes place:

- Internal UDD tables are created to store user IDs and permissions granted.
- The ownership of all tables is amended to be "**dba**".
- The user "**dba**" is created.

Only user **dba** can modify or drop system tables and indices. Non-dba users will be able to view these tables; with the exception of **UDDUSER**, which can only be viewed by user `dba`.

Only user **dba** can create and remove users; for example:

```
USER ken kenpwd
```

```
USER ken REMOVE
```

User **dba** can also modify user passwords; although, a non-dba user can change only his or her password. For example:

```
USER ken KENPWD
```

Note: All usernames however entered are converted to lowercase.

GRANT and REVOKE Syntax

The data dictionary system table **UDDAUTH** is maintained by the following commands:

```
GRANT privilege ON [ TABLE ] table_name TO { username,... } | PUBLIC
REVOKE privilege ON [ TABLE ] table_name FROM { username,... } | PUBLIC
```

where `privilege ::=`

```
{ ALL [ PRIVILEGES ] [ ( column_name,... ) ] }
| { SELECT [ ( column_name,... ) ] }
| DELETE
| { INSERT [ ( column_name,... ) ] }
| { UPDATE [ ( column_name,... ) ] }
```

and where PUBLIC implies ALL users.

These commands can be used using the UNIX and Windows U/SQL Interactive SQL Utilities, [usqli](#) and [Win U/SQLi](#) respectively.

For example:

- `revoke all on employee from public`
Remove all access privileges on table `employee` from all users.
- `grant select on employee to jim, bill, ken`

Grant SELECT access privileges on table employee to users jim, bill and ken.

- `revoke all (salary) on employee from jim`

Remove all access privileges on column salary within table employee from user jim.

- `grant update (salary) on employee to bill`

Grant UPDATE access privileges on column salary within table employee to user bill.

Only **dba** may specify access privileges on that table.

Table and column access privileges for a user are established in the following increasing order of priority:

| | |
|-----------------|--|
| Highest: | An entry for this user for this column within this table Entry for this column for PUBLIC Entry for this user for this table |
| Lowest: | Entry for this table for PUBLIC |

If no Grant/Revoke command has been applied to this table or column, then no additional restrictions will be imposed by Grant/Revoke security.

Note: *Setting a user to be Read Only via Host-Level security (LINK!) takes precedence over Grant/Revoke security.*

To ensure that data files are read efficiently, the user must have SELECT access privileges on the columns that make up the primary key of the table being viewed.

Even if not using a **SELECT** statement, it is necessary to have SELECT permission on any tables/columns used in any **WHERE** clause.

Batch Entry

In practice and for convenience, all the users and their passwords, that is **USER** statements, followed by the table and column privileges, that is **GRANT** and **REVOKE** statements, are placed in a text file.

This text file must have a '.SQL' extension, for example, **USERACC.SQL**, and its contents executed via the Interactive U/SQL, [Win U/SQLi](#) or [usqli](#), utilities. For example, after executing the UNIX server based **usqli** utility, enter the text file name at the U/SQL> prompt:

```
U/SQL> USERACC
```

Note: *The '.SQL' extension is not included.*

Great care must be exercised over the security of this file as it contains all the users' passwords and privileges.

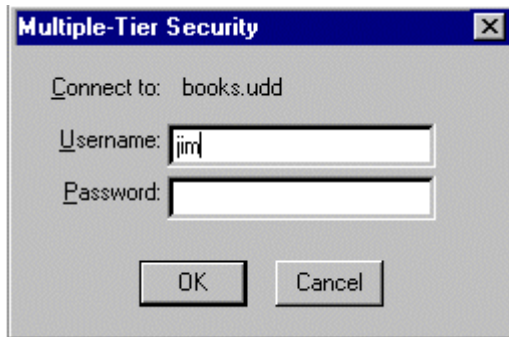
The same can be done through **Win U/SQLi**, loading and executing the SQL. Ensure that there is a statement delimiter ';' between each **GRANT** and/or **REVOKE** statement. Using a file to set up permissions is recommended so that the permissions can easily be set up again in the case that a new UDD is created.

Connecting to a Data Source

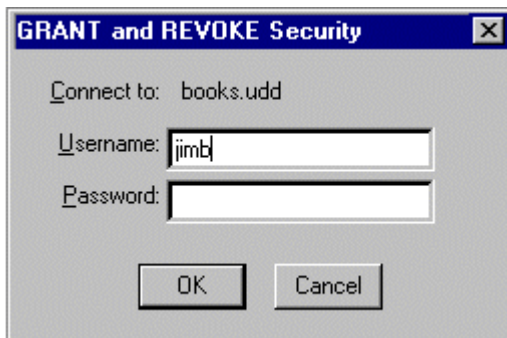
A data source is checked, at the time of connection, for the presence of GRANT and REVOKE security.

Connecting to a data source, in which GRANT and REVOKE security is active, causes U/SQL Adapters to prompt the user for a username and password. The user is offered a maximum of three attempts to supply a correct username and password. The previous username, if available, is supplied as a default when prompting for a username.

Connecting to a data source, in which Multiple-tier Security, see the section [Multiple-tier Security - User Connection](#), in addition to GRANT and REVOKE security, is active, causes U/SQL to prompt the user for a host username and password. This username and password combination, if correct, is automatically passed through to the GRANT and REVOKE security level in order to check if an equivalent entry is present within the list of defined users set up using the **USER** command listed above. If not, the user is prompted for a username and password in the same manner as described above.



Example Multiple-Tier Security login



Example GRANT and REVOKE Security login

GRANT and REVOKE (G&R) Security Issues

User Table 'Views'

When Grant/Revoke security is in effect, if a user has read-access to a subset of the columns of a table, a `SELECT * FROM <table>` command will return all the accessible columns. Any columns the user could not query explicitly will not appear. That is, the "*" operator in a SELECT statement evaluates to "all accessible columns" as opposed to "all columns".

Read Only

When a read-only data source is enforced, either by the directive `ReadOnly=Yes` or the Multiple-tier user security access `<user_name>=readonly`, read-only G&R is activated.

The G&R specific commands, `USER`, `GRANT` and `REVOKE` are disabled, but should they be used then the following message is displayed:

```
Cannot modify GRANT and REVOKE security within a read-only
data source.
```

Passwords

User passwords, held in the `UDDUSER` table in the `UDD`, are never displayed. For example, using the UNIX Interactive U/SQL utility, `usqli`:

```
U/SQL> select * from udduser;
USER PASSWORD
-----
dba *
fred *
```

Using U/SQL Manager

Once G&R security has been enabled, only the Database Administrator, that is the user `dba`, can update table(s) using the U/SQL Manager, for COBOL data sources.

SQLDriverConnect()

If you are creating applications that require G&R usernames and passwords to be processed, then this can be achieved by the ODBC function **SQLDriverConnect()**, which has been extended in U/SQL to include:

```
GR_UID=gr_uid;GR_PWD=gr_pwd;
```

where:

`gr_uid` G&R security username

`gr_pwd` G&R security password

An incorrect username or password results in an ODBC error code 28000 (Invalid authorization specification) and an error message text of:

```
GRANT and REVOKE Security
```

The function **SQLDriverConnect()** also supports as standard the Multiple-tier user connection username and password security. An incorrect username or password also results in an ODBC error code 28000 and the error message text for an unsuccessful attempt of this security is:

```
Multiple-Tier Security
```

For details on the **SQLDriverConnect()** function refer to the *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide*.

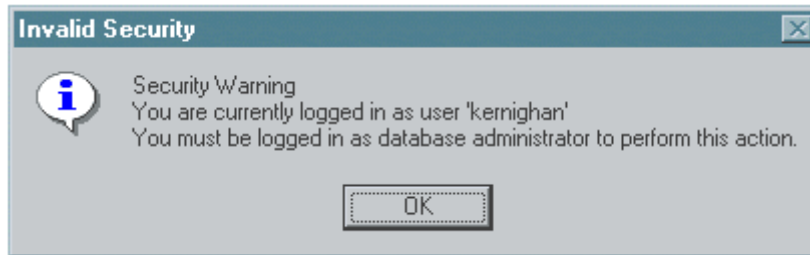
Export or Import

The following applies to the Export and Import facilities:

- An attempt by a non-`dba` user to export or import any table results in the following being displayed in the UNIX Interactive U/SQL utility, [usqli](#):

*** Error: User must be Database Administrator.

and the following error dialog is displayed in client based Interactive U/SQL utility, Win U/SQLi:



- While the list of users and permissions is contained within the tables UDDUSER and UDDAUTH respectively, it is not recommended to import or export these tables directly. Rather, it is recommended that a script be maintained containing the list of commands used to generate all the G&R permissions and for this to be used again in the event of a UDD having to be recreated.

If export or import is required, then note that only the dba may do this, G&R security must have been activated prior to import (by the command "USER dba <password>") and all user passwords must be set up separately as the import will not preserve the original passwords.

GRANT and REVOKE Error Messages

The following are error messages associated with GRANT and REVOKE security:

- GRANT and REVOKE security access denied for user "username"
- GRANT and REVOKE security not active
- User is neither the table owner nor the Database Administrator
- User must be Database Administrator
- Other than Database Administrator, user can only alter their own password
- Attempt to remove non-existent user "username" from table UDDUSER
- Maximum user name length of 20 exceeded
- Maximum password length of 20 exceeded
- Table table_name does not exist
- Column column_name does not exist
- Privilege* access denied on table: table_name
- Privilege access denied on column: column_name
- Privilege access denied on index: index_name.

Query Planning

Query Planner

The type of plans examined by the U/SQL Query Planner are referred to as nested-loop plans. All possible nested-loop plans for the tables of the query are examined and the cheapest to execute is chosen as the best plan. By cheapest we mean the one that uses the least estimate of disk I/O activity.

For example, consider a query based upon three tables **A**, **B** and **C**. One of the possible nested-loop plans could be represented as:

```

B
  A
    C
    
```

By this we mean that first table **B** is examined for all eligible rows, and for any such row table **A** is then examined for all eligible rows and for each such row table **C** is then examined for all eligible rows. At this stage for each eligible row of **C** we will have eligible candidate rows from **A** and **B** that meet any join criteria and hence a row of the result set of the query. Perhaps we can see why this is called the nested-loop method.

For the three tables **A**, **B** and **C** there are 6 possible nested-loop plans to consider:

```

A  A  B
  B  C  A
    C  B  C

B  C  C
  C  A  B
    A  B  A
    
```

In general for **n** tables there are **n!** (**n** factorial) possible nested-loop plans to examine. **n!** can get very large as **n** increases, for example:

| n | n! |
|---|-----|
| 1 | 1 |
| 2 | 2 |
| 3 | 6 |
| 4 | 24 |
| 5 | 120 |

| | |
|---|-------|
| 6 | 720 |
| 7 | 5040 |
| 8 | 40320 |

Luckily most queries fall into the lower range of number of tables. Also there are techniques which are used to limit examining lower branches of the possible set of plans where it is obvious some previously examined plan is already cheaper than the partially examined plan in hand (a so called branch-and-bound algorithm).

The whole rationale of examining the different plans is that some are cheaper to execute than others and one will be the cheapest. For any one of the $n!$ plans over n tables we need to find the cheapest method of executing that particular ordering of tables. Consider again the 3 table plan:

B

A

C

When costing this plan we start with table **B**. How should table **B** be accessed? We are looking for a way that uses the least I/O and delivers the minimum number of candidate rows. The Query Planner knows about the structure of **B** (size and indices) and also about any part of the **WHERE** clause that effects table **B**. In this case only parts of the **WHERE** clause that are independent of any other tables can be considered because **B** has no superior tables in the plan. Specifically this will be predicates that involve columns of **B** and literals or parameter markers. For example:

```
SALARY > 20000 and AGE BETWEEN ? and ?
```

If there is no part of the **WHERE** clause that can be used at this point there is no alternative but to access **B** sequentially and consider every row as a candidate row. It may be that part of the **WHERE** clause can be applied to **B** to filter the candidate rows but still a complete sequential scan of the table is required. This will have the same I/O cost but will render less candidate rows that will drive the remainder of the plan. Alternatively the applicable part of the **WHERE** clause may reference columns of **B** that form a significant part of some index on **B**. This can yield a much cheaper access method and also a dramatically reduced number of candidate rows. In the extreme, if all columns of a unique index are specified as equal to some value then the I/O cost is one index access and the number of candidate rows is one (or zero). With this method of analysis we can arrive at a decision on the cheapest way to access table **B**.

We can then move on to consider table **A**. Again we need to determine the part of the **WHERE** clause that is applicable at this point. This will be parts that refer to columns of table **A** OR columns of table **B** but NOT columns of table **C**. In general this requirement is referred to as pruning the predicate tree. Having determined the pruned **WHERE** clause that can be used we apply the same analysis as we did in the table **B** case. Specifically we can consider predicates of the form:

```
A.col1 = B.col2
```

Once the cheapest method of accessing table **A** is determined this cost is multiplied by the number of candidate rows for table **B**, as this access will have to be repeated that number of times. Similarly the estimate of the number of rows returned from **A** is multiplied to indicate how often table **C** will be accessed.

We then move on and examine how to access table **C**. This time any part of the **WHERE** clause can be used in the analysis.

The general form of this method of analysis is used in the U/SQL Query Planner. The end result is a plan which is defined in terms of:

- An ordering of the tables in the query
- For each table a method of access

For our example 3 table query this may be expressed as something like:

1. Table **C** using duplicate index 2
2. Table **A** sequentially with filter
3. Table **B** using unique index 0

This section discusses:

- [Viewing the Query Plan](#)
- [Affecting the Query Planner.](#)

Viewing the Query Plan

It is possible to view the query plan that the U/SQL Server engine evaluates based on the given SQL statement and details of the tables and their indices etc. The object of providing the query plan is to assist you evaluate whether you have specified the optimal SQL statement. Sometimes it is better to re-arrange the statement to make better use of available indices etc.

The query plan can be viewed via the Interactive U/SQL utilities, either the client-based [Win U/SQLi](#) or the UNIX server-based [usqli](#).

In Win U/SQLi, you are able to see the query plan in its own window by selecting the **Query Plan** option from the **Query** menu.

In usqli, at the U/SQL> prompt, set showplan 1:

```
U/SQL> showplan 1
```

Refer to the [usqli on UNIX Servers](#) section.

Take, for example, the following SQL statement, querying the UDD's system tables:

```
select uddtables.tabname, colname from uddtables, uddcolumns,
uddsources where source='BB' and uddtables.tabname =
uddsources.tabname and uddcolumns.tabid = uddtables.tabid;
```

Then the following query plan is produced:

```
(Query: 1) UDDCOLUMNS SEQUENTIAL
```

```
(Query: 1) UDDTABLES INDEXED
```

```
Index No. 0
```

```
Lower : Set UDDTABLES.TABID to UDDCOLUMNS.TABID
```

```
Upper : UDDCOLUMNS.TABID >= UDDTABLES.TABID
```

```
With Filter : UDDCOLUMNS.TABID = UDDTABLES.TABID
```

```
(Query: 1) UDDSOURCES INDEXED
```

```
Index No. 0
```

```
Lower : Set UDDSOURCES.TABNAME to UDDTABLES.TABNAME
```

```
Upper : UDDTABLES.TABNAME >= UDDSOURCES.TABNAME
```

```
With Filter : (UDDSOURCES.SOURCE = 'BB' and UDDTABLES.TABNAME =
UDDSOURCES.TABNAME)
```

The above query plan shows the order in which the tables (defined in the order 0, 1 and 2) are processed:

- UDDCOLUMNS SOURCES, table (1), is scanned sequentially because since there is no way to use an index for all three tables, the query planner has chosen to scan this table to get better index use on the other tables.
- For each column within UDDCOLUMNS, the equivalent table entry in UDDTABLES (table (0)) is determined using its unique key.
- Finally, table (2), UDDSOURCES is queried to retrieve the table source type, and this is checked against "BB". If this check is successful the row is a valid one.

Note: *This query is not optimal in many cases, because (unknown to the query planner) there are likely to be many more rows within UDDCOLUMNS than UDDTABLES or UDDSOURCES. See the next section, Affecting the Query Planner, to discover how the query planner's knowledge of the tables can be improved.*

Affecting the Query Planner

A simple way in which the performance of the Query planner can be enhanced, involves how the files are accessed. Other methods are highlighted in the section [Tuning the Query Planner](#).

The way in which the files are accessed can dramatically affect the performance of the query.

The Query Planner in the absence of any specific information assumes that each application table consists of 1000 rows requiring 100 disk pages to be read. It is possible to influence the Query Planner by updating any application table's details in the **UDDTABLES** table with the number of rows (column nrows) it contains.

This does not need to be an exact value. The Query Planner looks at the relative sizes between application tables in making its decisions. Sometimes two application files are nearly the same size, but the Query Planner takes the 'wrong' file as the driving file so you can 'force' the Query Planner to use the 'right' file by changing the number of rows appropriately

Here is an example of influencing the query planner. Take the following query:

```
select customer.cust_name, orders.odate, orders.ordno, oline.lineno,
stock.stock_title, oline.qty
from none.customer customer, none.oline oline, none.orders orders,
none.stock stock
where orders.custcode = customer.custcode and oline.ordno =
orders.ordno and stock.stknum = oline.stknum;
```

The default plan is

```
(Query: 1) OLINE SEQUENTIAL
```

```
(Query: 1) ORDERS INDEXED
```

```
Index No. 0
```

```
Lower : Set ORDERS.ORDNO to OLINE.ORDNO
```

```
Upper : OLINE.ORDNO >= ORDERS.ORDNO
```

```
With Filter : OLINE.ORDNO = ORDERS.ORDNO
```

```
(Query: 1) CUSTOMER INDEXED
```

```
Index No. 0
```

```
Lower : Set CUSTOMER.CUSTCODE to ORDERS.CUSTCODE
```

```
Upper : ORDERS.CUSTCODE >= CUSTOMER.CUSTCODE
```

```
With Filter : ORDERS.CUSTCODE = CUSTOMER.CUSTCODE
```

Transoft U/SQL User Guide

```
(Query: 1) STOCK INDEXED
Index No. 0
Lower : Set STOCK.STKNUM to OLINE.STKNUM
Upper : STOCK.STKNUM <= OLINE.STKNUM
With Filter : STOCK.STKNUM = OLINE.STKNUM
```

As with the query listed in the previous section, this query plan assumes that the files are all the same size, and as such is not optimal. The OLINE file is a detail file to the ORDERS master file. It is almost always better to read the master file in such circumstances first, and then read all detail records from that one.

The query planner initially assumes that all tables contain 1000 rows. This can be amended by adjusting the nrows field within the UDDTABLES table:

```
update uddtables set nrows=1000000 where tabname="OLINE";
```

Now the query planner knows that the OLINE table is much larger (it now contains a million rows), and the query plan would be amended as follows:

```
(Query: 1) ORDERS SEQUENTIAL

(Query: 1) CUSTOMER INDEXED
Index No. 0
Lower : Set CUSTOMER.CUSTCODE to ORDERS.CUSTCODE
Upper : ORDERS.CUSTCODE >= CUSTOMER.CUSTCODE
With Filter : ORDERS.CUSTCODE = CUSTOMER.CUSTCODE

(Query: 1) OLINE INDEXED
Index No. 0
Lower : Set OLINE.ORDNO to ORDERS.ORDNO
Upper : OLINE.ORDNO <= ORDERS.ORDNO
With Filter : OLINE.ORDNO = ORDERS.ORDNO

(Query: 1) STOCK INDEXED
Index No. 0
Lower : Set STOCK.STKNUM to OLINE.STKNUM
Upper : STOCK.STKNUM <= OLINE.STKNUM
With Filter : STOCK.STKNUM = OLINE.STKNUM
```

Note how the plan is now driven by ORDERS, rather than OLINE.

Tuning the Query Planner

The following directives can be used to tune the performance of U/SQL:

| Directive | DSD support | Description |
|---|--|---|
| ForceSimilarIndex Example | C-ISAM (If you plan to use this with other DSDs, contact Transoft support.) | Setting ForceSimilarIndex to Y allows you to force a change of index. For example, the following condition causes a change of index to be used for the query plan: If two tables exist in the query T1 and T2 , where: <ul style="list-style-type: none"> • T2 is using an index • T1 is scanning sequentially • T2 is using a variable from T1 to read • the variable being used forms part of an index in T1 In this example, the query planner changes to read along the index in T1 . This is designed to speed up read operations so that the rows for T2 are fed to it in approximately the correct order. The default for ForceSimilarIndex is N . |
| SetInIndexUse Example | All DSDs | Setting SetInIndexUse to Y allows you to optimize a "where <item> IN (subquery)" clause for index use. This allows jumps to avoid inefficiencies, for example when a subquery returns values near the first and last in the values, and therefore scans in all values between them. The default for SetInIndexUse is N . |
| AlternativeIndex Example | C-ISAM, ACUCOBOL, Micro Focus DSDs. | Setting AlternativeIndex to Y enables you to use overlapping indexes. For C-ISAM and COBOL, this is performed at the update dictionary stage, when a file UDDALTIND is populated. (If UDDALTIND is not present, you need to recreate the dictionary to enable it). During an update, all indices are examined to see if there are overlapping fields |

| | | |
|---|----------|--|
| | | <p>which could be used to map onto the existing index.</p> <p>When an index is split into sections around the record, this option will not pick them up as valid, because the logic of reassembling the key field would become too complex.</p> <p>The default for AlternativeIndex is N.</p> |
| <p>IndexStats Example</p> | All DSDs | <p>Setting IndexStats to Y allows the use of UDDSTATS to determine the choice of index. If UDDSTATS has not been set up, this has no effect. If UDDSTATS has been populated, information there is used to determine costing information for each index and part thereof.</p> <p>The default for IndexStats is N.</p> |
| <p>Distinct Jump Example</p> | All DSDs | <p>When you set DistinctJump to Y, using a query such as select distinct item from file, the engine can use the index on a column (item in this case), to jump past duplicate values.</p> <p>This directive applies to all DSDs. The default for DistinctJump is Y, that is, it is switched on by default.</p> |
| <p>CacheTables</p> | All DSDs | <p>Setting CacheTables to Y enables the query engine to remember the UDD structure of the table between statements. This can be useful if you are likely to be performing many small queries, on the same tables, on the same connection. (When the connection is broken, the table structure is forgotten.)</p> <p>This directive is only recommended in cases such as inserting many rows into one file where the same file is being used for each, or where repeated queries are being made against the same file, such as an internet-based query tool performing much the same query each time.</p> <p>The default for CacheTables is N</p> |
| <p>PartialIndex</p> | All DSDs | <p>Setting PartialIndex to Y makes use of an index even if you are not using the first part of the index in the query. It assumes that there are some fields in the query that are indexed.</p> |

| | |
|--|--|
| | <p>The default for PartialIndex is N.</p> <p>Note: <i>PartialIndex</i> will never make a difference unless statistics (IndexStats=Y) is set, because unless information is available to confirm that it would be preferable to use a subsidiary part of a key, it is too dangerous (it is far more likely to slow down a query than speed it up unless statistics are employed).</p> |
|--|--|

Worked Examples

This section contains worked examples of improvements that were outlined in the table above.

Prerequisites

To use the following examples, you must have the Books database installed. This is shipped with the Transoft U/SQL product. Each of the examples makes use of part of the Books database. For more information refer to the [Demonstration - Books Wholesaler](#) section.

Where the example refers to specific indices, these are the basic indices in the books database. Some data sources have more indices, but you can ignore these for the purpose of these examples.

Note: *To be able to use most of these performance enhancement settings, you must first understand the data, and have a good understanding of what the query plan means.*

You can write many of the following queries more simply (without subqueries, for example), but the following queries are valid, and demonstrate problems without being too complex.

Using SetInIndexUse

[Back to description](#)

In the following query:

```
select * from customer where custcode in (select distinct custcode
from salehist where svalue=399);
```

the subquery returns the values (C011, C016). The query plan looks like:

```
(0) CUSTOMER INDEXED
```

```
(Query: 1) CUSTOMER INDEXED
```

```
Index No. 0
```

```
Lower : Set CUSTOMER.CUSTCODE to the Lowest of Subquery 1.1
```

```
Upper : CUSTOMER.CUSTCODE <= the Highest of Subquery 1.1
```

```
With Filter : CUSTOMER.CUSTCODE =any Results of Subquery 1.1
```

```
(Query: 1.1) SALEHIST SEQUENTIAL
```

```
FILTER : SALEHIST.SVALUE = 399
```

The engine jumps to the lowest value returned by the subquery, and stops when it reaches the highest. However, this jumping can still be inefficient, especially where the subquery returns values near the first and last in the file, when the engine scans all the intervening space (in this example, the engine would read C011, C012, C013, C014, C015, and C016).

To improve on these inefficiencies, you can use the **SetInIndexUse=Y** directive to allow the engine to jump past the intervening gap. The query plan filter is amended to:

```
(0) CUSTOMER INDEXED
(Query: 1) CUSTOMER INDEXED
Index No. 0
Lower : Set CUSTOMER.CUSTCODE to the Lowest of Subquery 1.1
Upper : CUSTOMER.CUSTCODE <= the Highest of Subquery 1.1
With Filter : CUSTOMER.CUSTCODE =inindex Results of Subquery 1.1
(Query: 1.1) SALEHIST SEQUENTIAL
FILTER : SALEHIST.SVALUE = 399
```

The word `inindex` in the filter line causes the engine to act as follows:

1. Jump to C011. This record is checked in the subquery result set, and is acceptable.
2. Read C012. This record is not in the subquery. The next record in the subquery result set is C016, so the query jumps to C016.
3. Read C016 normally, followed by C017. At C017, the upper bound check fails and the query completes.

Using ForceSimilarIndex

[Back to description](#)

In the example query:

```
select a.custcode, b.custcode from customer a, customer b where
a.custcode = b.custcode;
```

the following query plan is produced:

```
(Query: 1) CUSTOMER SEQUENTIAL
(Query: 1) CUSTOMER INDEXED
Index No. 0
Lower : Set CUSTOMER.CUSTCODE to CUSTOMER.CUSTCODE
Upper : CUSTOMER.CUSTCODE >= CUSTOMER.CUSTCODE
With Filter : CUSTOMER.CUSTCODE = CUSTOMER.CUSTCODE
```

In this case, you want to read the two files in the same order, so that C001 is read out of file 'a' at the same time as it is read out of file 'b'. (This is especially important if customer is a very large file). For this reason, it uses the index, even though there is no condition in this example to imply that an index on file 'a' would be of use.

Using the **ForceSimilarIndex=Y** directive informs the engine not to perform a sequential scan, but instead, if the conditional field linking to the lower table is available as an index, to use that to read the file. In this example, `a.custcode` links to the indexed field `b.custcode`. The query plan therefore becomes:


```
(Query: 1) CUSTOMER INDEXED
Index No. 0
Lower :none
Upper :none
(Query: 1) CUSTOMER INDEXED
Index No. 0
Lower : Set CUSTOMER.CUSTCODE to CUSTOMER.CUSTCODE
Upper : CUSTOMER.CUSTCODE >= CUSTOMER.CUSTCODE
With Filter : CUSTOMER.CUSTCODE = CUSTOMER.CUSTCODE
```

With many data sources, this is a quicker than not using **ForceSimilarIndex**, because when C001 is read in the first file, the whole block is read into memory. Then, when the second file is read, this block is read into memory as well.

This directive is useful as long as the two files are running in parallel, because most of the data read operations come from a cache rather than being read in a random order. Obviously in the real world the data files will be much larger than this (CUSTOMER only has 17 records), and reading in the correct order rather than random access becomes much more significant.

This directive is recommended for C-ISAM, where the file handler always reads on an index even for a sequential scan. For most other data sources the file handler reads sequentially much faster than reading indexed, so setting this directive may in fact slow down queries.

Using Index Statistics

[Back to description](#)

Note: In order to use Index Statistics the directive **IndexStats** must be set to **Y**, for example, add **IndexStats=Y** to the **Data Source Defaults** section, of the **usqlsd.ini** configuration file on UNIX or using the U/SQL Administrator on Single-tier or Service Manager for Multiple-tier on Windows 2000 and Windows NT Server.

To determine which index to use for a file, it helps to know how effective the index is. For example, in the standard books database the table **salehist** has the following index:

SALEHIST_IX001, which contains the fields, **YEAR, PERIOD, CUSTCODE, STKNUM, SVALUE**.

The year is always 94. This means that using the year is of no real use, as you are likely to get back the entire dataset. When you include period, there are 6 distinct values (94/1, 94/2, 94/3, ..., 94/6). When you include custcode, there are 84 rows, and so on. If, for example, you have three parts of the index, you can expect a number of rows defined as NROWS / 84 to be returned, which in this case is 427/84, or approximately 5 rows.

As the dataset grows, you only need to generate these statistics if the data skews differently. For example, if there are 1000 customers in four regions, and the database grows to 10000 customers, but these customers are still in four regions, the statistics do not need to be rebuilt, as the increase is fairly uniform. However, if the salehist file grows from having only one year to having 10, that constitutes a major change, because it changes from selecting all the rows to having 1/10 of the rows against each value.

The statistics numbers are stored in a file called **UDDSTATS** in the database. The number of rows is stored, as before, in **UDDTABLES** under the column **NROWS**. The statistics numbers are broken down by tablename, index number, and part-of-index. For example, for **books.udd**, the statistics are as follows:

| tablename | indno | part1 | part2 | part3 | part4 | part5 |
|-----------|-------|-------|-------|-------|-------|-------|
| BUDGET | 0 | 1 | 4 | 16 | 192 | NULL |
| CUSTOMER | 0 | 17 | NULL | NULL | NULL | NULL |
| OLINE | 0 | 28 | 71 | NULL | NULL | NULL |
| ORDERS | 0 | 28 | NULL | NULL | NULL | NULL |
| SALEHIST | 0 | 1 | 6 | 84 | 406 | 427 |
| STOCK | 0 | 60 | NULL | NULL | NULL | NULL |

In each case, there is only one index per table, and in each case at the final part all the rows are accounted for, since these are all unique indices.

This is most useful where there are multiple indices on a table, when the query planner can examine the indices for their effectiveness. However, there can even be differences in query execution with one single indices. Take, for example, the following query:

```
select a.custcode, a.cust_name, b.svalue from customer a, salehist b
where a.custcode = b.custcode and b.year=94 and b.period=1;
```

It can be executed in one of two ways:

- Use year and period to select values in `salehist`, then use a unique key of `custcode` to access the customer file. Under the current statistics, this gives 427/6 rows for the first section (a cost of 69.5). For the second section, multiply the rows so far (69.5) by the expected rows at this level: 17/17 (1 row), giving a cost of 1*69.5 for this level. If you add this to the first level, this gives a cost of 139.
- Scan the customer file and then use year / period / `custcode` to access the `salehist` file. This has a scan of 17 records of the customer file, so the cost of level 1 is 17. Accessing the `salehist` file using year / period / `custcode` gives an expected 427/84 rows, or 5.08. If you multiply by expected rows again, you get 86.4. Added to the 17 for the first level, this gives a total cost of 103.

The query plan produced before running the **usqlstats** utility is:

```
(Query: 1) SALEHIST INDEXED
Index No. 0
Lower : Set SALEHIST.PERIOD to 1,
Set SALEHIST.YEAR to 94
Upper : (SALEHIST.YEAR <= 94 and SALEHIST.PERIOD <= 1)
With Filter : (SALEHIST.YEAR = 94 and SALEHIST.PERIOD = 1)
(Query: 1) CUSTOMER INDEXED
Index No. 0
Lower : Set CUSTOMER.CUSTCODE to SALEHIST.CUSTCODE
Upper : CUSTOMER.CUSTCODE <= SALEHIST.CUSTCODE
```

With Filter : CUSTOMER.CUSTCODE = SALEHIST.CUSTCODE

This example demonstrates that it is better to scan the first file sequentially, then to use a much better index, than to have an index for each file (including having a unique index for the second). This is often the case in a real-world situation, as it is often better to concentrate on using a very good index on the largest file in a query than to be concerned about scanning small ones.

The index statistics are also used for distinct jumping. For example, the query:

```
select distinct year from salehist;
```

expects only one row returned, so will always choose to use the index. However, the following example:

```
select distinct year, period, custcode, stknum from salehist;
```

does not use the index to jump, as it expects 406 rows, by the time jumping itself is added (performing an indexed read after each record), the time taken to use jumping would be greater than the time to scan the file.

To produce index statistics for the database:

1. First ensure you have the UDDSTATS file within the database. To determine if this is present, do:

```
select * from uddstats;
```

If this does not return a table, you must export, recreate and re-import the database. Before re-importing, ensure the directive **IndexStats=Y** is set.

2. Set up a DSN on a Windows system, which can access the data through Win U/SQLi.
3. Start a DOS window and run the usqlstats utility from either the U/SQL CD or downloaded from the FTP site (see Installing the usqlstats utility from the ftp site section). To run the usqlstats utility, type:

```
usqlstats books.udd
```

amending "**books.udd**" to whatever the data source name is under Windows. For more information see the usqlstats utility section.

After running the usqlstats utility on **books.udd** the following query plan is generated:

```
(Query: 1) CUSTOMER SEQUENTIAL
```

```
(Query: 1) SALEHIST INDEXED
```

```
Index No. 0
```

```
Lower : Set SALEHIST.CUSTCODE to CUSTOMER.CUSTCODE,
```

```
Set SALEHIST.PERIOD to 1,
```

```
Set SALEHIST.YEAR to 94
```

```
Upper : ((SALEHIST.YEAR <= 94 and SALEHIST.PERIOD <= 1) and  
CUSTOMER.CUSTCODE >= SALEHIST.CUSTCODE)
```

```
With Filter : ((SALEHIST.YEAR = 94 and SALEHIST.PERIOD = 1) and  
CUSTOMER.CUSTCODE = SALEHIST.CUSTCODE)
```

Using overlapping index fields

[Back to description](#)

In some complex data tables, there may be several overlapping fields. If these fields constitute an index, it is possible for one field to be selected, and the query planner not to know that it forms an indexable column. For example, the customer file:

```
create table CUSTOMER (
    CUSTCODE char(4),
    CUST_NAME char(30),
    CUST_REGION char(6)
);

create unique index CUSTIX on CUSTOMER (CUSTCODE);
```

is set up in **books.ufd** with fields as follows:

```
CISAM_FIELD(tabname, fldname, type, length, offset, ndec, ... )
"CUSTOMER", "CUSTCODE", "char", "4", "0", "0", ...
"CUSTOMER", "CUST_NAME", "char", "30", "4", "0", ...
"CUSTOMER", "CUST_Region", "char", "6", "34", "0", ...
```

If you add two additional overlapping fields:

```
"CUSTOMER", "CUSTCD2", "char", "2", "0", "0", ...
"CUSTOMER", "CUSTCD3", "char", "8", "0", "0", ...
```

which appear when a **select *** is performed, for example:

| CUSTCD3 | CUSTCD2 | CUSTCODE | CUST_NAME | CUST_Region |
|----------|---------|----------|----------------------|-------------|
| C001Alde | C0 | C001 | Alden & Blackwell | EAST |
| C002Book | C0 | C002 | Book Bargains Oxford | WEST |
| C003The | C0 | C003 | The Bookcentre | WEST |

CUSTCD2 is simply the first two characters of the CUSTCODE field, and CUSTCD3 includes the first four characters of the CUST_NAME field. Then, if you run the following query:

```
select * from customer where custcd3="C001Alde";
```

it would normally scan the database, since CUSTCD3 is not defined in the index (only CUSTCODE is). If alternative indices have been activated, the query is translated into:

```
select * from customer where custcd3="C001Alde" and custcode =
{fn left("C001Alde",4)};
```

You can use the additional part to perform an indexed search. This will work for more complex queries and indices, including multi-part indices.

To activate alternative indices, first check to see if the **UDDALTIND** table exists. To check if the **UDDALTIND** table exists, perform:

```
select * from uddaltind;
```

If you get an error, the table does not exist.

If the table exists, you only need to update the UDD.

If the table does not exist, you must export, recreate and re-import the database with the directive **AlternativeIndex** set to **Y**.

During the import process, all possible alternate indices are generated automatically, and stored in the **UDDALTIND** table. However, when you use an info command on a table, only genuine (not alternative) indices will appear, and the showplan shows which genuine index of the table is being used.

Distinct Jumping

[Back to description](#)

Distinct jumping is turned on by default.

Example 1

For example, to work out in which years sales have been made, use:

```
select distinct year from salehist;
```

The data at the start of the file is:

| YEAR | PERIOD | CUSTCODE | STKNUM | SVALUE |
|------|--------|----------|--------|--------|
| 94 | 1 | C001 | 005269 | 71.6 |
| 94 | 1 | C001 | 023021 | 484.5 |

In the first query, after the value 94 has been read, there is no point reading any more records of year=94. Because of this, the engine increments the value it is reading, to 95, and jumps to that point. In this example, there are no records with a year = 95, so the query ends.

The query has opened a file, read the first record, performed an indexed search, and stopped. In this way, the query performs two read operations rather than 427 (one for every record in the file).

Example 2

The following example is slightly more complex, for years and periods, using the same data:

```
select distinct year, period from salehist;
```

The same basic principle applies as in Example 1, but this time the second data item is incremented. In this way, when 94 1 has been read from the file, the period is incremented to 2, and the engine jumps to that value. It finds 94 2, and stores that as another record. It continues incrementing until 94 6, when the increment finds no more values (there is nothing beyond 94 7). In this case, there have been 12 read operations rather than 427.

Distinct jumping only applies when all the items being examined are present at the beginning of an index.

In the example:

```
select distinct cust_region from customer;
```

the customer file index is:

```
create unique index CUSTOMER_IX001 on CUSTOMER (CUSTCODE);
```

As `cust_region` is not contained within this index, the query planner performs the usual scan of the table.

In some cases, the query planner adds the distinctness. For example:

```
select min(year) from salehist;
```

Since only distinct values can affect this query, the planner effectively amends it to be:

```
select min(distinct year) from salehist;
```

From this point, the planner can use distinct jumping. This applies to both the min and max operators.

In a similar way, the following query finds all the queries for any year where customer C001 has ordered anything:

```
select * from salehist where year in (select year from salehist where custcode='C001');
```

The subquery performs identically, regardless of whether it has multiple values or distinct values, that is, whether it returns ('94') or ('94', '94', ...). You can therefore translate the query into:

```
select * from salehist where year in (select distinct year from salehist where custcode='C001');
```

Once again, the engine can jump in the subquery.

Query Planner Hinting

In U/SQL Revisions 3.10.400 and above, you can use hints in the [Query Planner](#). You do this by placing hints immediately after the **SELECT** statement.

There are two types of query planner hinting. They are:

| | |
|--|--|
| <code>/** INDEX(table,index) */</code> | To enforce the use of a specific index when accessing table. |
| <code>/** ORDERED */</code> | To join the tables in the order supplied, and not to look for a better plan. |

You must place these hint comments immediately after the **SELECT** statement, but you can intersperse them with additional comments.

The index can be referred to by number or name.

The old style of comment, using '#' to comment out the remainder of the line, can also be used, immediately followed by a '*' to form a hint-style comment.

Query hinting applies to all DSDs.

Using Query Hinting

Transoft recommends that you make use of the performance optimizations in this manual as far as you can. After you have implemented the optimizations, they are activated automatically for all relevant queries.

However, there may be circumstances where your knowledge of the database contents makes a particular query plan obvious to you, but not to the query planner. In these circumstances, you can direct the query planner, as shown below, to perform the table in a particular way.

Note: *The query planner will never override the user, so a badly-hinted query may be far worse than no hinting at all.*

Specifying the order of tables

To dictate the order of the tables in a query plan, you can add the `/** ordered */` comment to the query. For example:

```
select /** ordered */ table1.* from table1, table2, table3;
```

In this example, the search begins at `table1`, then searches `table2`, then searches `table2`. As with all hints, you can use lower, upper or mixed case.

Specifying the index to use

To dictate which index is used in a query, you can use the `/** index */` hint. For example:

```
select /** index(salehist, salehist_ix001) */ year, period from salehist;
```

In this example, `salehist_ix001` is the index name obtained from an **info** command. This example directs the query planner to consider no other index for use on this table, just to use the index specified.

To direct a table to scan sequentially, using no index at all, you can use the following format, which specifies none as the index to use:

```
select /** index(salehist, none) */ year, period from salehist;
```

Specifying the order of tables and which index to use

You can combine the instructions to specify the order of tables, and which index to use. For example:

```
select /** ordered index(table1, index1) index(table2, index2) */ *  
from table1, table2;
```

These instructions can be in any order, but must be located directly after the select statement.

Sample Applications

Demonstration - Books Wholesaler

A demonstration of an application for a Books Wholesaler is supplied with the U/SQL Client software. This chapter describes what is included in the Books Wholesaler demonstration.

Single-tier

Windows

On the PC, under the U/SQL Client software installation directory (by default, **C:\Program Files\USQLC**), there is the directory **BOOKDEMO**. This contains three subdirectories, **DATA**, **BIN**, and **SOURCE**. These subdirectories contain the following files:

| | |
|---------------|--|
| DATA | Contains the application data files and the UDD, booksw.udd for the supplied data source driver, which are used to run the Books application. |
| BIN | Contains the executable books32.exe (a Visual Basic application) and an Access database, books32.mdb , which can also be used to run the Books application without using U/SQL . |
| SOURCE | Contains the Visual Basic source to the Books application. |

Multiple-tier

UNIX

The relevant data files and the UDD, **books.udd** are contained on the host in an **example** directory, below the base directory of the U/SQL Server software installation (by default, **/usr/usqls/example**).

Windows NT Server

The relevant data files and the UDD, **books.udd** are contained on Windows NT Server, in the directory **BOOKDEMO**, below the base directory of the U/SQL Server software installation (by default, **C:\USQLCS\BOOKDEMO**).

Windows

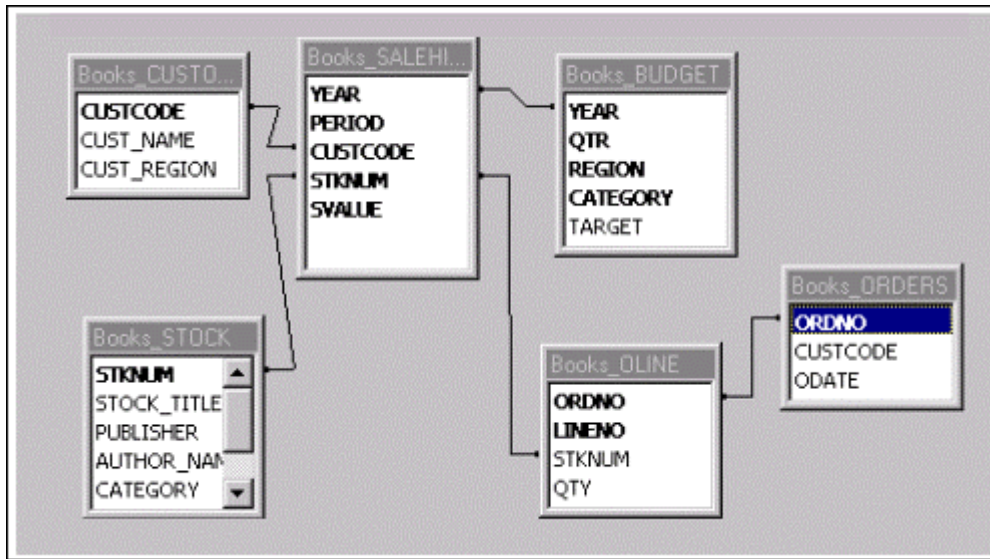
On the PC, under the U/SQL Client software installation directory (by default, **C:\Program Files\USQLC**), there is the directory **BOOKDEMO**. This contains two subdirectories, **BIN** and **SOURCE**. These subdirectories contain the following files:

| | |
|------------|--|
| BIN | Contains the executable books32.exe (a Visual Basic application) and an Access database, books32.mdb , which can also be used to run the Books application locally on the client |
|------------|--|

| | |
|---------------|--|
| | platform without involving the U/SQL Server. |
| SOURCE | Contains the Visual Basic source to the Books application. |

Sample Data

The following diagram, using Microsoft Access, shows the simple data structure for the Books Wholesaler example. These are the six files or tables in the system. This is the logical view of the data with the lines drawn between the tables showing the logical joins:

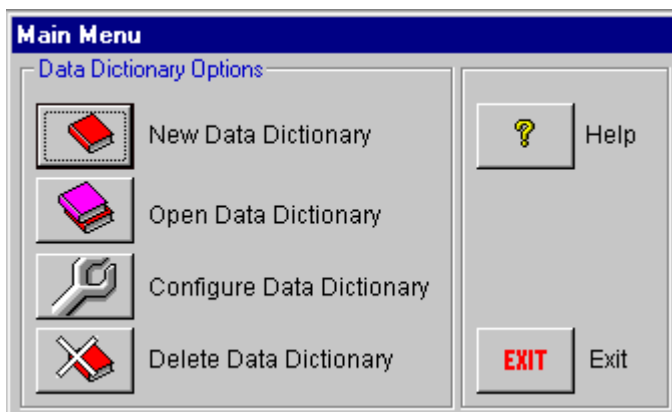


Note: Only the primary data connections are included to improve clarity. The UDD contains details of these tables and their fields or columns.

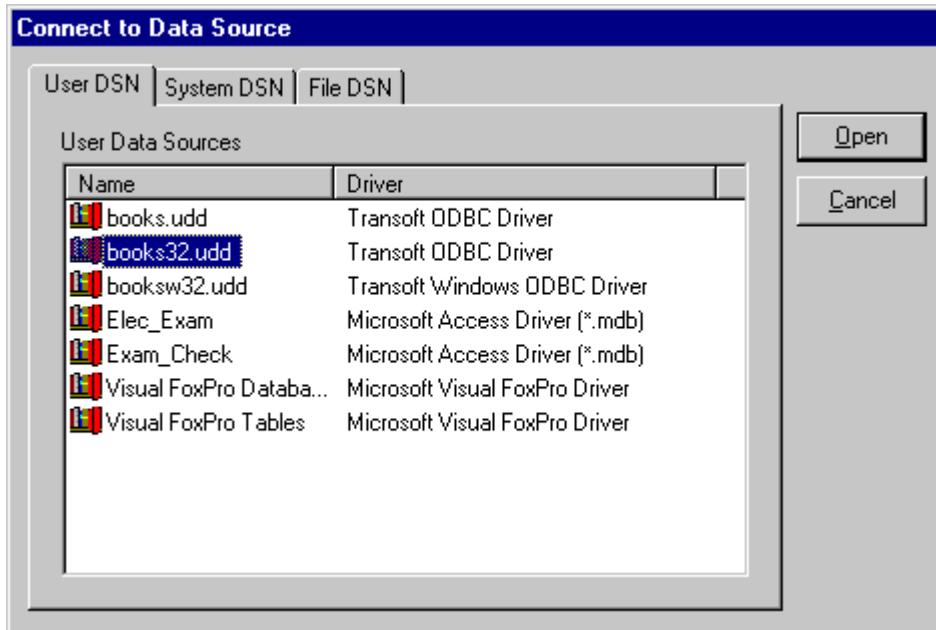
View the UDD for COBOL Data Source

If you have been supplied with a U/SQL Manager for your COBOL data source, you can view the contents of the books UDD (Single-tier: **booksw32.udd**; Multiple-tier: **books32.udd**).

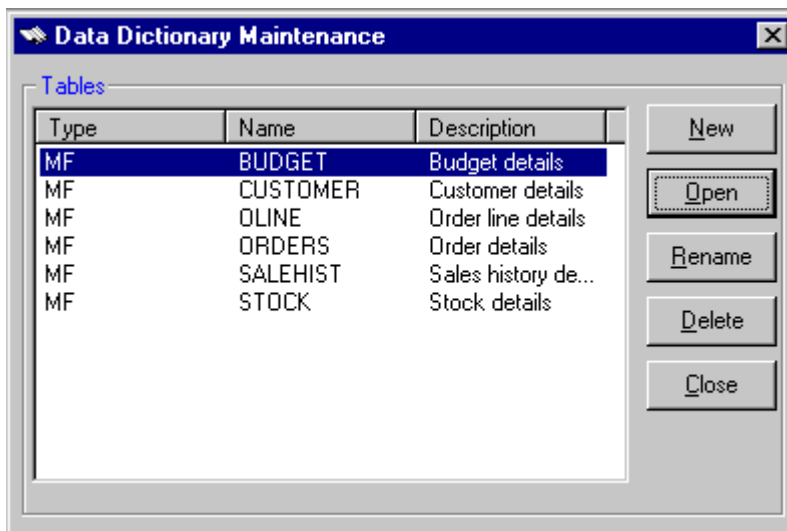
From the **Start** menu double-click on the U/SQL Manager icon in the U/SQL Client program group. The **Main Menu** is displayed:



Click **Open Data Dictionary**. The **Connect to Data Source** dialog box is displayed:

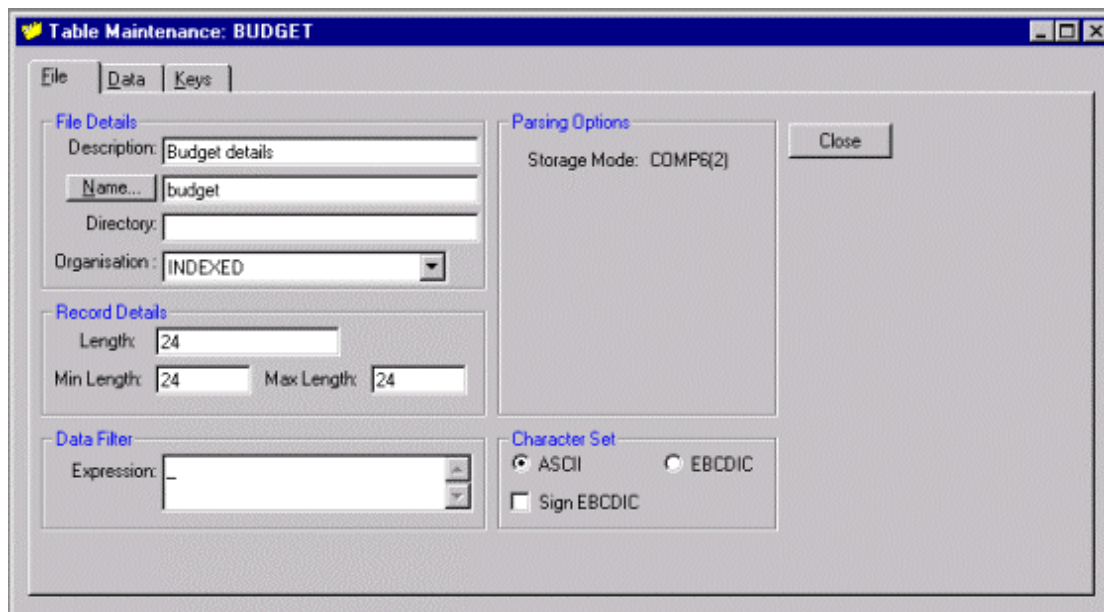


Select the **books32.udd** data dictionary and click **Open**. The **Data Dictionary Maintenance** dialog box is displayed:



This shows the Tables available from the UDD.

To view the details of any Table, either select it and then click **Open** or double-click the table name. The **Table Maintenance** dialog box for that table is displayed:



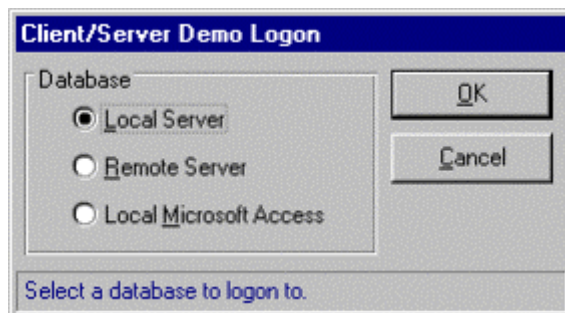
You can then view the **File**, **Data** and **Keys** information. Refer to the [Planning to Use U/SQL Manager](#) section for details on how to use the U/SQL Manager.

Running the Books Demonstration Application

Ensure you have installed the U/SQL Client and Server software successfully and, for Multiple-tier, that the U/SQL Server is running.

To run the Books Wholesaler demonstration application, double-click on the **Books** icon from within the U/SQL Client program group.

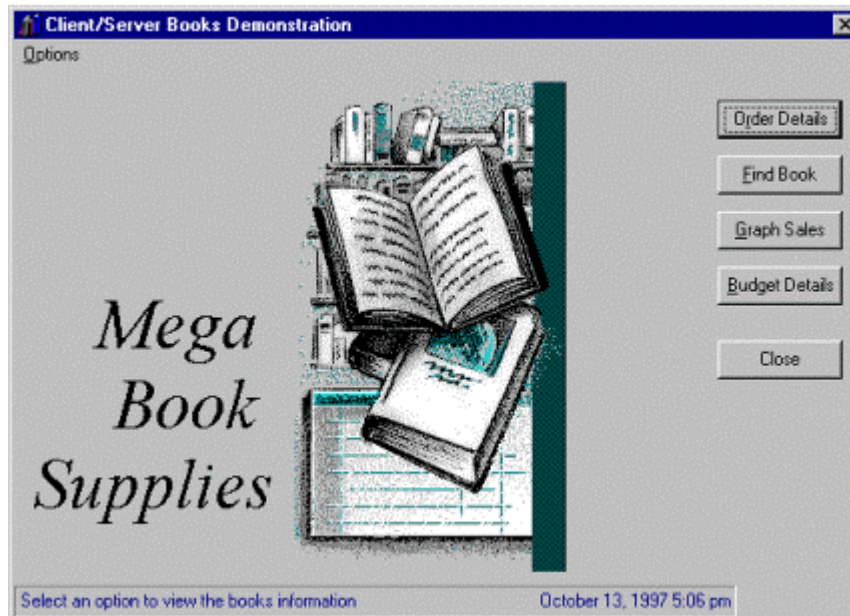
The **Client/Server Demo Logon** dialog box is displayed:



This contains the following options:

- Local Server (if you have a Single-tier version of U/SQL Adapters)
- Remote Server (for example, on a UNIX or Windows NT Server host)
- Local Microsoft Access (using a Microsoft Access database making no use of U/SQL Adapters)

After making the appropriate selection, the following main form menu is displayed:



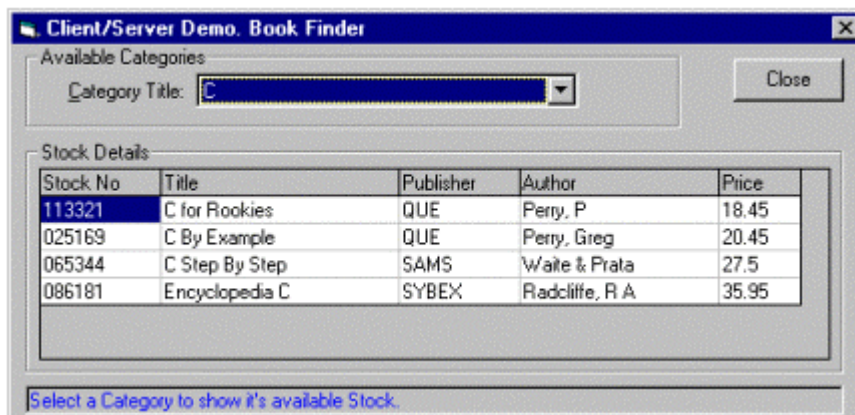
Select the first item by clicking the **Order Details** button. The **Client/Server Demo Order Display** dialog box is displayed:

| Line | Stock No | Title | Qty | Unit Price | Total |
|------|----------|-------------------------|-----|------------|--------|
| 1 | 082053 | Easy Lotus 1-2-3 2.4 | 9 | 18.45 | 166.05 |
| 2 | 024594 | Second Complete AutoCAD | 11 | 15.95 | 175.45 |
| 3 | 006581 | 10 Min. Gde Excel 4 | 10 | 9.95 | 99.50 |
| 4 | 065344 | C Step By Step | 6 | 27.50 | 165.00 |
| 5 | 054797 | ABCs of Excel 4 | 14 | 17.95 | 251.30 |

Enter an Order Number to view the Customer and Order details.

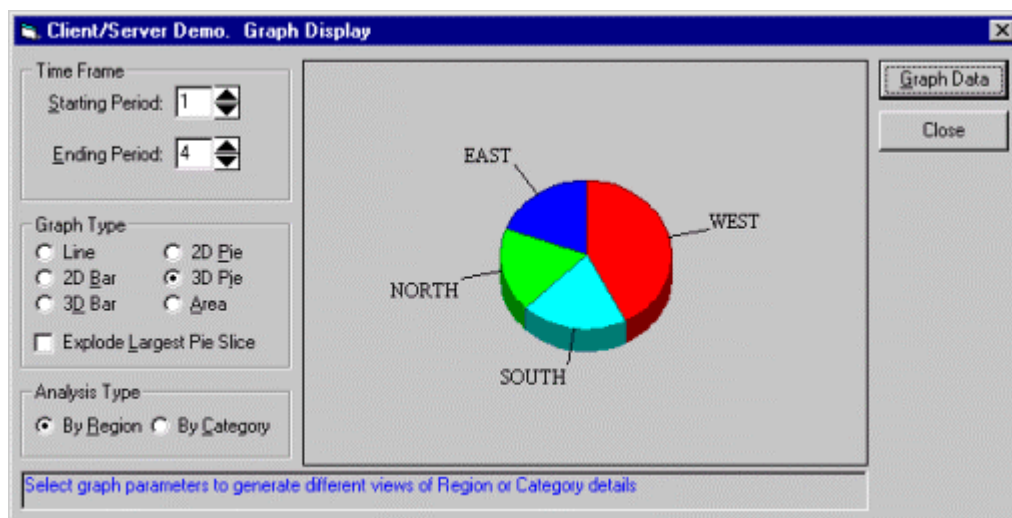
This demonstrates a four-way join between the **ORDERS**, **OLINE**, **STOCK** and **CUSTOMER** files or tables. Any order number in the range 123001 to 123027 can be entered. After entering the order number click **OK**.

For the second example, click the **Find Book** button on the main form menu. The **Book Finder** dialog box is displayed:



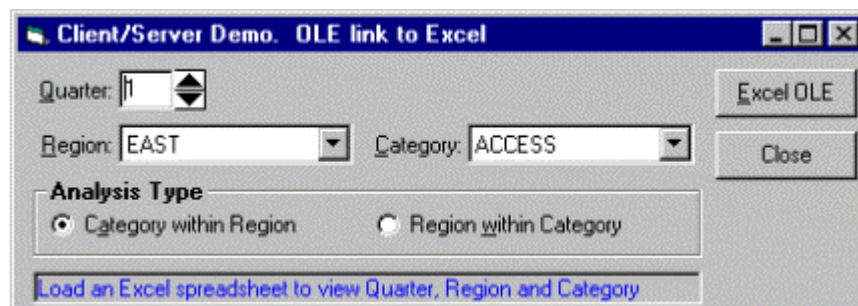
As the application is invoked, the **Stock Details** section of the dialog box is filled with all the categories of book that the wholesaler sells. Selecting any one of the categories, for example, 'C for Rookies', results in the individual book title details for that category being displayed from the STOCK file.

For the third example, click the **Graph Sales** button on the main form menu. The **Graph Display** dialog box is displayed:



You can select the number of periods you require, by region or by category of book, and you can experiment with various forms of graph. This provides an Executive Information System in 'real time' for non-relational data.

For the last example, click the **Budget Details** button on the main form menu. The **OLE link to Excel** dialog box is displayed:



You must have Microsoft Excel 4, or higher, on your PC for this part of the demonstration. Make your selection for a particular quarter, the actual region or book category from the combo boxes and the Analysis Type (either Category within Region or Region within Category), then click the **Excel OLE** button. The application then invokes Microsoft Excel and, using OLE and a macro, populates a

spreadsheet via an SQL SELECTION of the data.

Sample SQL Queries

Refer to the [Querying and Manipulating Data](#) section for examples of SQL syntax for querying and manipulating data for ODBC 2 compliance. The examples shown make use of the Books Demonstration data provided with your data source. In general, you will be able to reproduce these examples yourself.

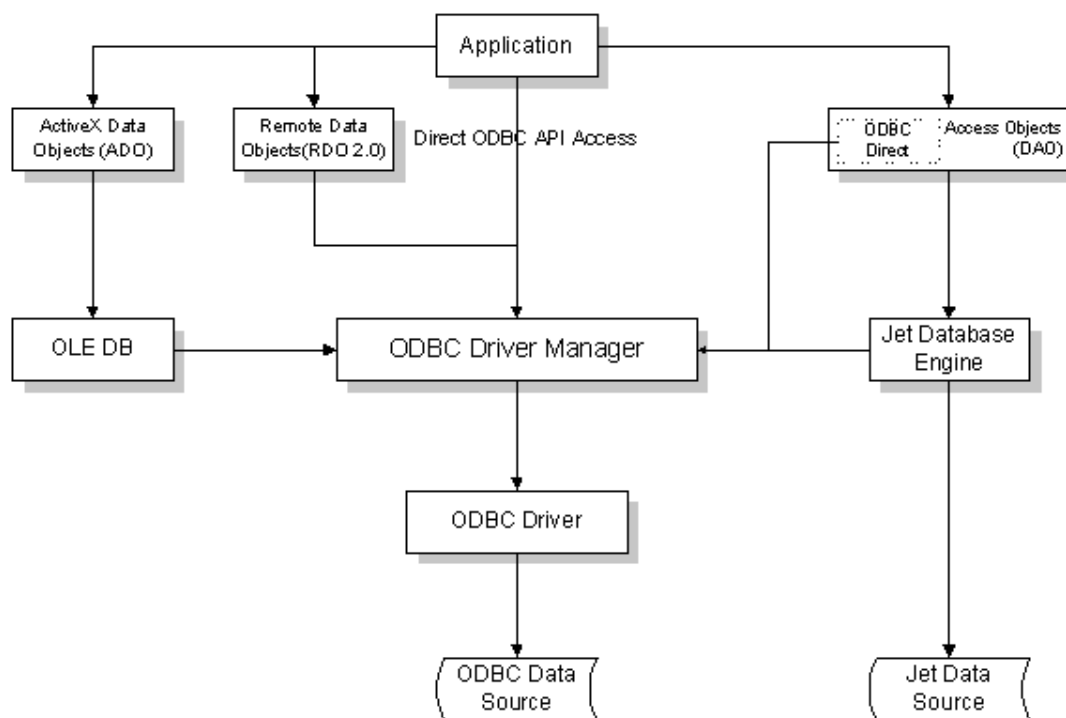
View Source of the Demo

Windows

On the PC, under the U/SQL Client installation, there is the directory **BOOKDEMO**. This includes the sub-directory SOURCE, which contains the Visual Basic source to the Books application. You can use this to see how the various functions have been implemented. In order to take advantage of this you must have a development copy of Microsoft Visual Basic 5 Professional installed on your PC.

Writing Applications Using U/SQL To Access Your Data

Revision 3.00 of the U/SQL ODBC drivers include improved support for some of the 32-bit data access methods available from Windows RAD tools such as Microsoft Visual Basic. The following diagram illustrates the different methods available for accessing data:



Data Access Options for Windows Developers

Data Access Objects (DAO)

Data Access Objects are available to Visual Basic and Visual C++ developers and Microsoft Office applications, and represent the easiest and most common method of accessing data from Microsoft applications. DAO uses the Microsoft JET engine, used by Microsoft Access, which makes it particularly effective when used to retrieve data from an Access database as well as an ODBC data source.

However, although it is possible to use DAO to connect to and retrieve data from ODBC data sources, this is not a particularly effective route to take. As the previous diagram shows, in this scenario the JET engine will still be being used, which is inefficient as it can often be duplicating processing which is either being performed by the ODBC driver manager or the ODBC driver itself.

This situation was improved when Visual Basic 5.0 was released, as it offered an improvement to connecting to ODBC data sources with DAO, called ODBC Direct. When this property is set (as the Type argument in the CreateWorkspace method) it prevents the JET engine from performing any processing and just implements the DAO object model on top of the ODBC API. This results in thinner client applications and faster query execution, as the JET engine no longer needs to be loaded or used, however there will be the loss of some JET specific functionality when using ODBC Direct - for a full description please consult the Microsoft Visual Basic 5.0 Books Online documentation.

Example DAO Code

A code example from Visual Basic 5.0:

```

Dim wsJET As Workspace
Dim wsODBC As Workspace
Dim m_daoDB As Database
Dim m_daoRS As Recordset
Dim m_daoField As Field
Dim szConn As String

' Create one workspace of each type
Set wsODBC = CreateWorkspace("ODBCWorkspace", "admin", "", dbUseODBC)
Set wsJET = CreateWorkspace("JETWorkspace", "admin", "", dbUseJet)

' Append workspaces to collection
Workspaces.Append wsODBC
Workspaces.Append wsJET

'N.B. JET database & ODBCdirect connection objects are
interchangeable
szConn = "ODBC;DSN=BOOKSW32.UDD;"

' Open an ODBCdirect connection
Set m_daoDB = wsODBC.OpenConnection("ODBCDirect", dbDriverComplete,
False, szConn)
Set m_daoRS = m_daoDB.OpenRecordset("Select * from customer",
dbOpenSnapshot)

For Each m_daoField In m_daoRS.Fields
    ' Runs through columns in recordset
Next

m_daoRS.Close
m_daoDB.Close

' Open a JET connection
Set m_daoDB = wsJET.OpenDatabase("BOOKSW32.UDD", dbDriverComplete,
False, szConn)
Set m_daoRS = m_daoDB.OpenRecordset("Select * from customer",
dbOpenSnapshot)

For Each m_daoField In m_daoRS.Fields
    ' Runs through columns in recordset
Next

m_daoRS.Close
m_daoDB.Close

```

Remote Data Objects (RDO)

Introduced with the release of Visual Basic 4.0 (32-bit) (Professional & Enterprise editions only), RDO offers easy and high-performance access to ODBC data sources. In much the same way as when using DAO with ODBCdirect, as described in the previous section, RDO is just a thin layer that operates directly above the ODBC API and does not use JET at all. RDO has extensive support for cursor libraries and an event model for trapping data events as they occur.

RDO is available only to developers using Visual Basic Professional or Enterprise editions - Revision 1.0 of RDO was introduced with Visual Basic 4.0; Revision 2.0 of RDO is available with Visual Basic 5.0. Revision 2.0 has relaxed the requirements for ODBC conformance from revision 1.0's requirements, which means that U/SQL Adapters is now compatible with RDO revision 2.0, subject to certain conditions mentioned in the section Notes on RDO, below.

Example RDO Code

A code example from Visual Basic 5.0: (ensure you have included Microsoft Remote Data Objects 2.0 in your project references)

```
Dim m_rdoMainEnv As rdoEnvironment
Dim m_rdoConn As rdoConnection
Dim m_rdoRes As rdoResultset
Dim tmpField As rdoColumn

Set m_rdoConn = New rdoConnection
m_rdoConn.CursorDriver = rdUseOdbc
m_rdoConn.Connect = "DSN=booksw32.udd;"
m_rdoConn.EstablishConnection

Set m_rdoRes = m_rdoConn.OpenResultset(szSQL,
rdOpenForwardOnly)

For Each tmpField In m_rdoRes.rdoColumns
    ' This code loops through each column in the resultset
Next
```

Notes on RDO

The important points to note in the above code are:

- **Cursor Type** - Note that the CursorDriver property of the connection is set to rdUseODBC before the connection is opened. This is important because RDO supports quite a rich set of cursor drivers and properties that are not implemented in the U/SQL ODBC Drivers, so RDO must know that the ODBC driver manager will be implementing them for this connection.
- **Recordset Type** - Note the setting of the final parameter of the OpenResultset call. The rdOpenForwardOnly setting opens a forward-scrolling recordset, which is the default recordset used by RDO. If your recordset needs to be updateable or have unrestricted movement then the U/SQL ODBC driver has been tested with rdOpenKeyset based recordsets, which should provide all the functionality required.

ActiveX Data Objects (ADO)

ADO is Microsoft's interface to OLE DB data access. OLE DB allows access to relational and non-relational data, wherever it may reside. Theoretically, data may be queried from such sources as ODBC data sources, word processor documents and e-mail messages. ADO can be used from RAD tools such as Visual Basic 5.0, but is also designed to be used in Web pages, such as Active Server Pages, over the Internet / Intranet.

When connecting to an ODBC data source OLE DB utilises the 'Microsoft ODBC Provider' to bring in the ODBC driver manager and use the standard ODBC interface to access the data.

At the time of writing the latest revision of ADO & OLE DB is 1.5, and is shipped as part of Microsoft's Data Access Components. U/SQL Adapters ODBC Drivers have been tested with this revision and, subject to certain conditions mentioned in the section Notes on ADO below, function with this revision.

Example ADO Code

A code example from Visual Basic 5.0 (ensure that you have included Microsoft ActiveX Data Objects in your project references)

```
Dim m_ADOConn As ADODB.Connection
Dim m_ADORes As ADODB.Recordset
Dim tmpField As ADODB.Field

m_ADOConn.Mode = adModeReadWrite
m_ADOConn.IsolationLevel = adXactChaos
m_ADOConn.Open "DSN-BOOKSW32.UDD", ""

Set m_ADORes = New ADODB.Recordset
szSQL = "SELECT * FROM customer;"
m_ADORes.Open szSQL, m_ADOConn,
    adOpenForwardOnly,
    adLockPessimistic

For Each tmpField In m_ADORes.Fields
' This code loops through each column in the resultset
Next
```

or a more concise method, in an Active Server Page for example, would be:

```
Set RS = CreateObject("ADODB.Recordset")
RS.Open "select * from stock;",
    "DSN=BOOKSW32.UDD;UID=admin;PWD=",
    adOpenForwardOnly,
    adLockPessimistic
```

Notes on ADO

The important points to note about the above code are the final two parameters on the recordset's Open method; these are the CursorType and LockType of the recordset being opened. The U/SQL ODBC Drivers only fully support forward scrolling recordsets, so it is important to explicitly set the CursorType parameter to adOpenForwardOnly when opening a recordset. The LockType setting is less important, but adLockPessimistic or adLockReadOnly are the only settings recommended for use with the U/SQL ODBC drivers.

Positioned Updates. Attempting to edit and update an ADODB.Field object from an open recordset will generate an error; all updating of data through ADO using U/SQL Adapters should be performed by executing SQL rather than by updating Field objects.

ODBC API

All of the previous methods of connecting to ODBC data sources, DAO, RDO & ADO have provided object models and functionality that ultimately calls the raw ODBC API functions. If you wish to achieve maximum performance from your application then you can call ODBC API functions directly from your application.

The benefits of calling ODBC functions directly is that you have full control over exactly what code is executed against the ODBC driver, and you have no code layers between your application and ODBC, so you should achieve the best possible performance for your application. The drawback to developing using direct ODBC API calls is that it requires a great deal more code from the application to implement even the most basic functionality and error handling.

Choosing The Data Access Method

Some of the interface data access methods discussed are only available to certain development tools; RDO is only available with the higher end editions of Visual Basic, and if you are developing a 16-bit Visual Basic application you will not have RDO or ADO available to you at all. Delphi provides its own database engine, which you can use to connect to ODBC data sources with, or you can use direct ODBC API calls.

However, assuming you have the choice of the four options discussed; DAO, RDO, ADO and ODBC API then you will need to choose one method over another, as they are not particularly interchangeable - although you can re-use connections between RDO and ODBC API calls if you wish.

The order in which the methods were discussed was, in our opinion, in order of ease-of use. For example both DAO and RDO have visual data source components that other controls such as grids can be 'bound' to; in this manner you can implement a fully-functional database 'front-end' with hardly any application code. In this respect there is little difference between DAO and RDO, although DAO provides a richer object model for the developer because of the JET engine.

At the time of writing, ADO has a visual component for Web pages, but not for use in Visual Basic, so an application written using ADO would need to implement the front-end functionality itself. An application written in pure ODBC API calls would need code to implement both the ODBC functionality and the front-end functionality.

The DAO, RDO, ADO and ODBC API order is also applicable in relation to the amount of development time required for implementing an application using each method, which is also related to the relative ease-of-use of each method.

In most cases the extra development time needed for writing with the ODBC API probably outweighs the performance increases you gain from using it, especially as RDO offers all the performance benefits of direct ODBC access together with the bound controls of DAO. DAO's ODBCdirect option allows developers to improve the performance of legacy code by simply changing the types of workspaces their application creates, rather than having to redevelop using direct ODBC access.

Probably the most important consideration is the scenario in which your application is going to be used; RDO is designed for a client-server situation, where the client is kept as lightweight as possible and the server does all the work. ADO further extends this model by providing for web-based clients and multiple relational and non-relational data sources. DAO, on the other hand, is the native Access engine which also happens to be able to connect to ODBC data sources; this is useful if you need to combine a local Access database with data from an ODBC data source for any reason, but otherwise it is not the best method of connecting to ODBC data.

Applications which use DAO or RDO will need less application code, resulting in a smaller executable, but will require more components at run-time - whereas ODBC API applications will tend to be larger but require fewer run-time components.

General UDD Information

Overview

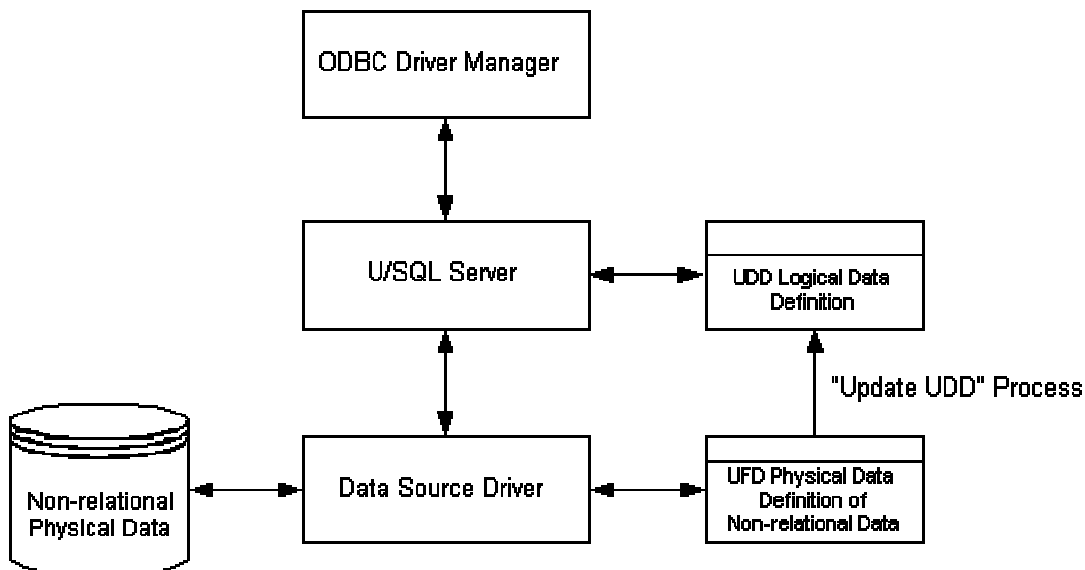
This section provides an overview of the dictionary technology and the steps involved in creating a Universal Data Dictionary (UDD) for U/SQL. Subsequent sections provide details on [how to set up a UDD](#) for the following non-relational data sources:

- [C-ISAM](#)
- [Transoft's Business Basic ISAM](#)
- [Transoft's U/FOS hierarchical database management system](#)
- [COBOL](#).

Note: Other data source drivers are continually being added to U/SQL, so for a complete list of data sources supported, contact [Transoft](#).

This section also covers [Expression Handling](#) and its syntax. For example, to differentiate between multiple record types within the same physical file, as separate logical tables, you are able to specify an expression that uniquely distinguishes each record type from the others.

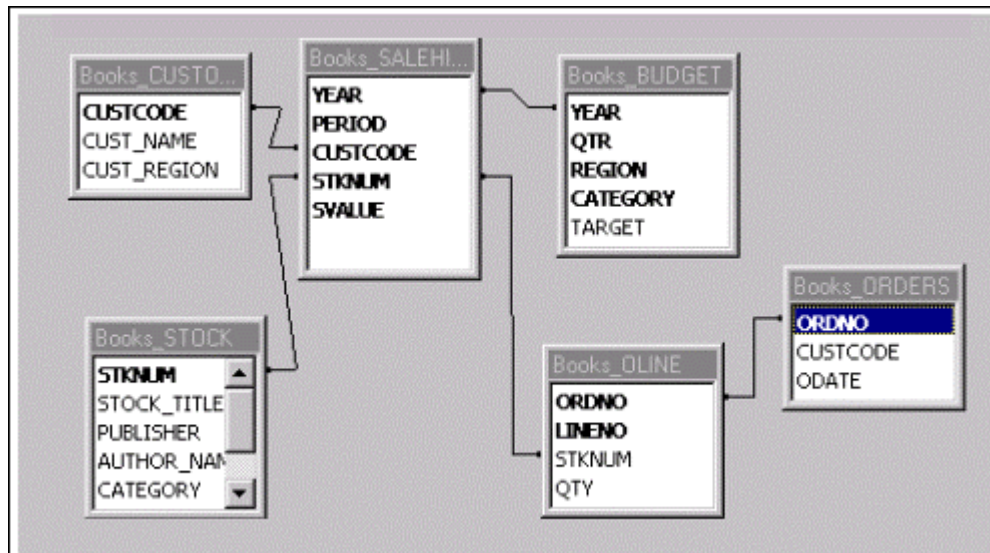
U/SQL employs a two-level data dictionary technology, to provide a 'relational view' of your non-relational data.



The UDD contains a set of system tables that provides a relational description of your data. It also contains the Universal File Dictionary (UFD), which is a further set of tables that describes the physical structure of the files being accessed.

Examples of UDD system tables are **UDDTABLES** and **UDDCOLUMNS**. Examples of UFD system tables are **CISAM_TABLE**, **CISAM_FIELD**, **CISAM_INDEX**.

The following diagram shows a simple data structure for the [Books Wholesaler Demonstration](#). These are the six tables in the system. This is the relational view of the data with the lines drawn between the tables showing the logical joins:



Note: Only the primary data connections are included to improve clarity.

The UDD contains details of these tables and their column names. The UFD includes details of:

- What the physical files are called on disk, for instance, the **CUSTOMER** table may be called **CUST.DB**.
- What the files contain in terms of fields, their data types and byte offsets.

ODBC-enabled products can only attach to what they believe to be relational tables and access them via SQL. Hence, these products see only the relational view of the U/SQL dictionary.

You can set up multiple UDDs, as you may want different users of the ODBC-enabled products to have different views of the same data. For example, you may not want some users to be able to view a payroll system. Alternatively, with a single UDD you can use GRANT and REVOKE security to exclude certain users from the payroll data tables and or particular columns. For example, the salary column. Refer to the [GRANT and REVOKE Security](#) section.

Creating a UDD

To create a new **UDD** for your application, you first describe the physical structure of your data files and their content in a **UFD** text file, for example, **dictionary.ufd**. These descriptions in this file are imported into the UFD part of an 'empty' dictionary, which is then updated to automatically create the 'relational view' to complete the dictionary.

U/SQL supports an ever-increasing number of non-relational data sources, for example, C-ISAM, Business Basic ISAM, U/FOS hierarchical database management system, and so on. Each non-relational data source has differing:

- File structures and access methods. For example, hierarchical, ISAM, sequential, direct, linked list.
- Indexing. For example, single index, multiple indexes, duplicates allowed.
- Record structures, including multiple records in the same file, differentiated by a record type field or expression.
- Data types of fields, for example, for COBOL, 'C', Business Basic, and other specifics such as the handling of dates and 'nulls'.

The amount of UFD information necessary to describe any non-relational data source is dependent on its complexity. For example, U/FOS being a hierarchical database management system is much more complex to describe than Business Basic ISAM or C-ISAM.

The following sections discuss:

- [Creating a Text File](#)
- [Comma Delimited UFD File](#)
- [Creating the UDD from the UFD Text File](#)
- [Amending the Data Dictionary.](#)

Creating a Text File

The data dictionary, for C-ISAM, Business Basic ISAM or U/FOS, is specified by creating a text file, for example, **dictionaryname.ufd**. It consists of a number of sections, each of which relates to a table in the UFD. Each table is specified as an entity, and a table type can be repeated as any number of entities. A table entity consists of the following:

```
table_name1(colname_1,colname_2,...)
-----
R1C1  R1C2  R1C3
R2C1  R2C2  R2C3
# Comment.
R3C1  R3C2  R3C3
*
```

- The first line contains the name of the table followed by the names of the columns delimited by commas and enclosed in parentheses.
- The next line contains a template which describes the position and maximum length of the data for each column. The format of this template is one or more groups of dashes, where the position and length of each group specifies the location and maximum length of the following data.

The amount of white space between the groups of dashes is not significant.

- There then follows one or more rows of data to be inserted into the UFD table. The row data must match the positions specified in the template.
- '#' in the first column of a line marks the beginning of a comment.
- '\' in the last column of a line signifies it continues on the next line.
- Separate tables are delimited by an asterisk '*' in the first column on a line.

Comma Delimited UFD File

Some of the lines in the UFD text file can become very long, particularly if many index key components are required, making it difficult to manipulate. To overcome this a comma delimited form of the UFD file can be specified.

This is achieved by removing the template and specifying each field item for each row of data separated by commas. String values must be enclosed in double quotes, "".

The special characters still apply; '#', comments character, '\' continuation character and '*' end of table character.

An example table:

```
table_name1(colname_1,colname_2,...)
R1C1,"R1C2",R1C3
12345,"Joe Smith",5678.76
# Comment.
R3C1,"R3C2",R3C3
*
```

Creating the UDD from the UFD Text File

Once you have created the **UFD** text file that describes the physical structure of your data files and their content, you proceed to create the **UDD** as follows:

- For Multiple-tier UNIX platforms, you create your UDD on the server using the Interactive U/SQL utility, [usqli](#).
- For Multiple-tier versions on Windows NT Server and for Single-tier versions (if available), you create your UDD using the Interactive U/SQL utility, [Win U/SQLi](#), on a client PC.

UNIX

On Multiple-tier UNIX platforms, once you have created the text file containing the entries defining the UFD structure of your data files, for example, **demo.ufd**, you must go through a three-step process to create the UDD:

1. In the **bin** directory below the base directory where you loaded the U/SQL Server software, for example **/usr/usqls/bin**, create an empty UDD, for instance, **demo.udd**, by running the Interactive U/SQL utility [usqli](#), with the **-c** switch:

```
cd /usr/usqls/bin
./usqli -c demo.udd
```

Note: *The dictionary must have a .udd extension.*

2. Next, you must load your UFD data, **demo.ufd**, into the UDD. This is achieved by running **usqli** again with just the name of the UDD:

```
./usqli demo.udd
```

and then, at the 'U/SQL' prompt:

```
U/SQL> import demo.ufd
```

```
U/SQL> quit
```

3. Finally, you must create the UDD, from the UFD information, by running **usqli** with the **-u** switch:

```
./usqli -u demo.udd
```

The message: All tables successfully updated, is displayed.

There are conditions where you do not get this message, for example, due to a missing key component in an index. To determine which table has the problem and what it is, interrogate the log file. Refer to the section How to Query the Log File in the Configure & Use manual.

or you can update one table at a time:

```
./usqli -u demo.udd AREAS
```

```
./usqli -u demo.udd COUNTIES
```

```
./usqli -u demo.udd CUSTOMERS
```

The UDD, **demo.udd**, is now ready for use.

To list the tables just loaded into the UDD, use the [tabs](#) command.

```
./usqli demo.udd
```

and then, at the 'U/SQL' prompt:

```
U/SQL> tabs
```

```
.
```

```
. [your table names are listed]
```

```
.
```

```
U/SQL> quit
```

Note: Ensure you perform the update step (3), otherwise the UDD is left in an incomplete state and is unusable.

Single-tier and Windows NT Server

If you have a Single-tier installation, or if you have a Multiple-tier version where your U/SQL Server software is installed on Windows NT Server, then you use the Interactive U/SQL utility, [Win U/SQLi](#) to create your UDD. Win U/SQLi is supplied with all U/SQL Client software installations.

Once you have created the text file containing the entries defining the UFD structure of your data files, for example, **demo.ufd**, you go through the following process to create the UDD:

1. If you have installed Multiple-tier U/SQL on Windows NT Server, ensure that the U/SQL Server is running.
2. Invoke **Win U/SQLi** on your client PC where you have the **demo.ufd** UFD text file.
3. Create a new UDD, for example, **demo.udd**, to which the UFD text file will be imported, by either clicking the **Create a new UDD** icon from the toolbar, or selecting **New UDD** from the **File** menu. Select where you

want the UDD to be created, that is **Local** for Single-tier and **Remote** for Multiple-tier.

You will then set up the data source within the U/SQL Administrator. (For Single-tier see the [U/SQL Administrator](#) section for more information. For Multi-tier see the [U/SQL Administrator Facilities](#) section for more information.)

4. Click the **Import Tables** icon or select **Import** from the **Table** menu. Select the UFD text file you have created, in this case, **demo.ufd**.
5. An **Import Tables** dialog box is displayed, showing each table as it is imported. Each successfully imported table has **Imported** displayed to the left of the table name.
6. After all the tables have been imported, click **OK**.
7. Next click the **Update selected tables** icon, or select **Update** from the **Tables** menu. Select the appropriate data source from the dialog box, and click **Update**. This creates the UDD tables from the UFD tables.
8. The message: *All tables successfully updated*, is displayed.
There are conditions where you do not get this message, for example, due to a missing key component in an index. Interrogate the log file to obtain which table has the problem and what it is. Refer to the [How to query the Log File](#) section.
9. Click **OK**.
10. You can check that the new UDD is correct by performing one or more queries on it using [Win U/SQLi](#).

Note: *Ensure you perform the update step 7, otherwise the UDD is left in an incomplete state and is unusable.*

Amending the Data Dictionary

If any amendments are required then either:

- The existing UDD must be deleted, the relevant changes made to the '.ufd' text file and the steps repeated, in the above section, to create and load the new UDD, or
- If adding a new data file, a UFD can be set up simply for that table and imported on its own. If amending a table in any way, it is recommended that the whole dictionary be recreated from the amended file. It is also recommended that backups be taken of the UFD and UDD files prior to any amendment.

Expression Handling

If you have multiple record types in the same physical file, you need to decide whether you want to describe each record type as a separate logical table, independently accessible via ODBC-enabled products. If you do, then you also need to know how each record type can be distinguished from the others. This is usually determined by one or more fields having particular values, known as differentiating values.

When you subsequently access each logical table via SQL, the U/SQL Server will automatically use the differentiating values or expressions to return only the appropriate rows (records) from the physical file.

IMPORTANT: *If the file uses arrays, the data items which determine record type cannot be within the repeating group, since this is evaluated AFTER the record type is determined.*

U/SQL includes an expression handler that accepts these differentiating values or expressions to determine the appropriate record type to be selected.

For example, consider an **EMPL** employee file which has two record types – the Employee Master Record and the Employee Salary Record. You could create two logical tables, one called, say **MASTER_EMPL** and the other **SALARY_EMPL**. The two record types are distinguished by the **REC_TYPE** field being "M" or "S" respectively. Thus for the **MASTER_EMPL** table you insert the expression:

```
REC_TYPE="M"
```

and for the **SALARY_EMPL** table the expression is:

```
REC_TYPE="S"
```

Note: *If you have multiple records or structures, in the same physical file, you must define each one as a separate logical table.*

This section covers the following topics:

- [Expression Syntax](#)
- [Logical Operators](#)
- [Comparison Predicate](#)
- [BETWEEN Predicate](#)
- [LIKE Predicate](#)
- [Pattern Syntax](#).

Expression Syntax

The U/SQL expression handler conforms to the X/Open SQL syntax for the **WHERE** clause search-condition that qualifies the selection of query rows.

Search conditions are one or more predicates, combined with the logical operators **AND**, **OR** and **NOT**.

The names of columns (fields) in expressions are NOT case-sensitive.

Note: *Sub-queries and ODBC functions within the search-condition are not supported.*

Logical Operators

The order of precedence among the logical operators is **NOT**, followed by **AND**, followed by **OR**. The order of evaluation at the same precedence level is from left to right. Parentheses can be used to change this order.

Comparison Predicate

A comparison predicate compares two values and has the form:

expression_1 comparison_operator expression_2

where *comparison_operator* can be any of the following:

- = equal to
- <> not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

The result is true or false, depending on the outcome of the comparison.

BETWEEN Predicate

A BETWEEN predicate tests whether a value is within a range of values and has the form:

expression_1 [NOT] BETWEEN *expression_2* AND *expression_3*

The predicate (without NOT) is equivalent to:

expression_1 >= *expression_2* AND *expression_1* <= *expression_3*

Using the keyword **NOT** negates the result in the manner of the NOT logical operator.

LIKE Predicate

A LIKE predicate compares a column value with a pattern and has the form:

column_name [NOT] LIKE *pattern_value*

The column *column_name* must reference a character string column, and *pattern_value* is a character string literal.

The result of the predicate is true or false depending on whether or not the value of the column referenced by *column_name* conforms to the specified pattern. Using the keyword **NOT** negates the result in the manner of the **NOT** logical operator.

Pattern Syntax

Within the **character string literal** represented by *pattern_value* in the above example, characters are interpreted as follows:

- The underscore character '_' stands for any single character.
- The percent character '%' stands for any sequence of zero or more characters.
- All other characters stand for themselves.

For example:

- `LIKE '%X%'` is true for any column value that contains the character X.
- `LIKE 'Y_'` is true for any column that is two characters wide and starts with the character Y.

Note: The **character string literal** can be bounded by either single quotes (') or double quotes (").

Handling of Data Arrays

Handling of Data Arrays

Virtually every computer language supports data arrays, and it is not unusual for legacy systems to store arrays in the data files. There is therefore, a need to be able to map the contents of an array into the relational 'table/column' model.

This section describes the various issues that can arise, and explains the options for mapping the data array in the UDD and the entries required in the Universal File Dictionary (UFD) textual file to generate the functionality.

Although there are variations in the support for mapping data arrays in the various U/SQL data source drivers, COBOL, U/FOS and so on, the principles involved are common. The approach taken in this section is to define and illustrate the issues in a way that will provide you with a level of understanding of the various options.

Note: Refer to the sections [Manually Create a COBOL UDD](#) and [Creating a UDD](#) for details on creating UFD textual representations of your data.

Terminology

Some documentation uses the term 'table of data' instead of 'data array'. This section uses the term 'data array' so that there is no confusion over the use of the word 'table' which has specific meaning in the 'Relational' context.

A **data array** can consist of one or more repeating elements. In the simplest case the element can be a single character, therefore by implication a 'string' of characters is stored in a 'character array'. However, in most cases, we would consider the 'string' of characters as a single entity - we would map the 'column' onto the storage for the entire string. The most common arrays are, therefore, repeating elements of integers (number storage), strings (character storage) or even structures (groups of fields).

In many cases the array is 'single-dimensional', in which case a single subscript (index) is necessary to access the different elements. However, it is also possible to address data storage in the form of a 'multi-dimensional' array, requiring two or more subscripts (index or sub-index) to address each element.

With U/SQL, it is the Data Source Driver that determines how the data is accessed, and although the data source driver may be linked to a language product, it is often possible to mix modules written in different computer languages and dialects in one application. For the purpose of simplicity and clarity, the COBOL syntax and terminology is used in the following text and illustrations.

Mapping Array Elements to a Relational Model

Data arrays are common components of business data. You define data arrays by including the **OCCURS** clause in the COBOL data description entry. For example:

```
03 SALES-TARGET PIC 9(8) OCCURS 3 TIMES.
```

There are two fundamentally different ways that these three fields can be mapped onto a 'relational' table.

- Consider each field as a separate column.
- Consider each field as a single column in separate rows.

For example, if the data array was used so that the first element was for sales in the USA, the second element for Europe and the third element for the rest of the world (ROW), it would be possible to map this data array onto three columns. This is known as a **Flattened array**, as it describes what happens when a COBOL **OCCURS** is flattened out into individual unique fields.

| USA_Target | Europe_Target | Rest_Target |
|------------|---------------|-------------|
| 1385 | 794 | 286 |
| 1245 | 2341 | 1860 |
| etc. | etc. | etc. |

One problem with this mapping technique is that if we add the columns we get a total for each region, when perhaps we want to obtain an overall total.

To do this, we could map the data into a two column model by introducing a subscript column (Region). Each original data record results in three rows in the new table. This is known as a **Repeating Group**.

| Region | Target |
|--------|--------|
| USA | 1385 |
| EUROPE | 794 |
| ROW | 286 |
| USA | 1245 |
| EUROPE | 2341 |
| ROW | 1860 |
| etc. | etc. |

There are several permutations and variations of this simple model of array handling:

- **Parallel OCCURS**

This is where two or more data arrays occur in the data record, and the corresponding elements in each array are treated as part of the same repeating group.

- **Nested OCCURS**

This is where two or more subscripts have to be indexed to access the elements of, what is, a multi-dimensional array.

- **OCCURS depending on**

This is where the number of elements in the data array can vary from record to record depending on the value stored in a controlling field.

There are certain record layouts that can look difficult to model; for example it is possible to imagine a combination of data arrays where the elements correspond in an inverted way; the first element of one array corresponds to the last element of another array. By making use of a temporary table in the UDD, it is usually possible to model complex layouts.

The following sections describe the various types of array in more detail:

- [Flattened Array](#)
- [Repeating Groups](#).

Flattened Array

This is where OCCURS are flattened out into individual unique fields.

Consider the following example of the ADDRESS that occurs three times:

```
01 DATA-RECORD.
   03 CUST-ID PIC X(4).
   03 CUST-NAME PIC X(15).
   03 ADDRESS PIC X(30) OCCURS 3 TIMES.
```

For this type of data you would probably want to flatten the OCCURS fields into three individual fields, one for each address line.

This can be done by considering the record as being:

```
01 CUSTOMER.
   03 CUST-ID PIC X(4).
   03 CUST-NAME PIC X(15).
   03 ADDRESS.
       05 LINE-1 PIC X(30).
       05 LINE-2 PIC X(30).
       05 LINE-3 PIC X(30).
```

Because the data storage of the original file description is identical to this revised model, it is simply necessary to use the correct field offsets when setting up the UFD textual file definition.

If this table was entered in a UDD, then the following query will produce the table below:

```
select fldname, offset, len from <xxx_field> where tabname =
"CUSTOMER";
```

Note: The value of xxx in the <xxx_field> table is the name of the UDD internal table that describes the layout of the data fields within the record, see below.

| fldname | offset | len |
|-----------|--------|-----|
| CUST_ID | 0 | 4 |
| CUST_NAME | 4 | 15 |
| LINE_1 | 19 | 30 |
| LINE_2 | 49 | 30 |
| LINE_3 | 79 | 30 |

You can identify the internal table name represented by **tabname** from within your UDD by either reference to the UFD text file used to create the UDD or by the following query:

```
select tabname from uddtables where tabname like "%FIELD" and
tabname not like "GEN%";
```

The above example was based on a UDD that supported the **COBOL_FIELD** internal table. Similar results are possible from other data source drivers (DSD).

Note: The U/FOS DSD uses slightly different syntax. The corresponding query is:

```
select fldname, picture, offset from ufos_field where tabname
= "CUSTOMER";
```

Repeating Groups

Consider the following example COBOL FD with the JOBS-DATA group occurring four times:

```
01 JOBS-RECORD.
    03      JOBS-KEY.
        05  JOBS-NO      PIC X(6) .
        03  JOBS-NAME    PIC X(15) .
        03  JOBS-DESCN   PIC X(20) .
        03  JOBS-DATA    OCCURS 4 TIMES.
            05  JOBS-ACCD  PIC S9(10)V99.
            05  JOBS-PAID  PIC S9(10)V99.
```

When the record is converted to form a relational table, the JOBS-DATA appears as four rows with columns of JOBS-ACCD and JOBS-PAID. The other fields, JOBS-NO, JOBS-NAME and JOBS-DESCN being repeated for each row.

First consider this record mapped into a flattened array:

| fldname | offset | len |
|------------|--------|-----|
| JOBS_NO | 0 | 6 |
| JOBS_NAME | 6 | 15 |
| JOBS_DESCN | 21 | 20 |
| ACCD_1 | 41 | 12 |
| PAID_1 | 53 | 12 |
| ACCD_2 | 65 | 12 |
| PAID_2 | 77 | 12 |
| ACCD_3 | 89 | 12 |
| PAID_3 | 101 | 12 |
| ACCD_4 | 113 | 12 |
| PAID_4 | 125 | 12 |

Now consider a temporary UDD table:

```
create temp table JOBS ( NO char(6), NAME char(15), DESCN char(20),
ACCD decimal(12,2), PAID decimal(12,2));
```

Note: For the creation and use of temporary tables in the UDD, refer to the section *Create Tables in UDD*.

Now populate this temporary UDD table using the following commands:

```
insert into JOBS
select jobs_no, jobs_name, jobs_descn, accd_1, paid_1 from
jobs_record;
```

```
insert into JOBS
select jobs_no, jobs_name, jobs_descn, accd_2, paid_2 from
jobs_record;
```

```
insert into JOBS
select jobs_no, jobs_name, jobs_descn, accd_3, paid_3 from
jobs_record;
```

```
insert into JOBS
select jobs_no, jobs_name, jobs_descn, accd_4, paid_4 from
jobs_record;
```

If you then query this temporary UDD table:

```
select * from jobs;
```

you will see it contains the columns in the Repeating Group form.

Note: *You may want to add an extra column into the table to indicate the element being used. For example, say the first element is 'North' in your application:*

```
create temp table JOBS (NO char(6), ID char(6),..... etc.;
insert into JOBS select jobs_no, "North", jobs_name ..... etc.;
```

Whilst this method is a very powerful way of processing some very difficult data layouts it is not possible to update the data in the original file system as the final query is made against the temporary extract. A more elegant solution is available with the versions of U/SQL that support COBOL data structures.

This is achieved by making use of two array mapping system tables in the UDD. You can check that these tables are available in your Data Source Driver (DSD) by using the command:

```
select tabname from uddtables where tabname like "%ARRAY%";
```

If support is available in your DSD, then there should be two rows returned, one for the object system table and one for the level system table.

The following five step process is the recommended way of setting up the parameters for mapping a Repeating Group table:

Step 1

Note: *Refer to the sections [Manually Create a COBOL UDD](#) and [Creating a UDD](#) for details on creating UFD textual representations of your data.*

The recommended first step is to map the required fields in the data record to columns in the table as though the data held in the array was not required. Remember that this requires the offset to fields after the array to be computed correctly; you cannot ignore the size of the array when computing the offset.

Where the UFD **xxx-field** table, for example, **cobol_field**, requires a sequence number, that is the column name is called **sequenceno**, then make sure you leave large enough gaps in the number sequence for inserting extra lines later.

It is worth checking that the UFD text file entries you have created will build into a valid and working UDD before proceeding to the next step.

To achieve this refer to the sections [Creating the UDD from the UFD Text File](#) and [Manually Creating a COBOL UDD](#).

Step 2

The next step is to add extra entries into the UFD **xxx-field** table to map the field(s) in the first element of the array into the appropriate position in the table. Remember that you have changed the number of entries in the **xxx_field** table and you need to adjust the counter in the associated **xxx_table** entry.

Note: As in [step 1](#), it is recommended that you check the UFD text file entries you have created will build into a valid and working UDD before proceeding to the next step.

Step 3

The basis of the **Repeating Group** array processing method is to define an internal field that can act as a **subscript**. When this type of array processing is operational, the Data Source Driver reads a record from the file and then the record is broken down into 'n' rows by indexing this subscript field to traverse the array(s).

Where the **xxx_field** table supports a sequence number, the entry should be inserted in the table immediately before the first array entry, from [Step 2](#), using an appropriate sequence number to retain the ascending sequence. Where the concept of a subscript 'object' is used by the Data Source Driver, for example, for U/FOS, the entry should be immediately after the other entries, usually the Data objects, to ease maintenance by keeping associated records together.

This **subscript** field is defined in the **xxx_field** table as follows:

| cobol_field | ufos_field | New entry required |
|-------------|------------|--|
| ----- | objname | Cross reference name, say SUB, to entry in ufos_object of objtype = 'SUBSCRIPT'. |
| ----- | fldno | Identify field within object (1 in this case as this is first and only subscript). |
| tabname | ----- | Table name. |
| fldname | fldname | An unique identifier (say IDX). |
| cobolname | ----- | Not used. |
| flddesc | ----- | Not used. |
| sequenceno | ----- | A sequence number of the entry in the table. |
| type | ----- | Set = 3 (Numeric). |
| usage | ----- | Set = 10 (Display). |
| ----- | fusage | Set = "DISPLAY". |
| signed | ----- | Set = 34 (Unsigned). |
| len | ----- | Set = 4. |
| offset | offset | Set = 0. |
| ndec | ----- | Set = 0. |

| | | |
|---------------|--------------------|--|
| picture | picture | Set = "9(4)". |
| levelno | ----- | Set to level of array in structure. |
| numdigits | ----- | Set same as 'len' column, that is, 4. |
| dateformat | ----- | Not used. |
| fieldflags | ----- | Set to 9 (bit 1 = 'data' and bit 4 = 'subscript' flags). |
| arraylevname | ----- | A unique identifier. |
| arraylevel | ----- | 'Depth' of array (1 in this case). |
| nullvalue | ----- | Not used. |
| colassignexpr | ----- | Not used. |
| scaling | ----- | Not used. |
| | ufos_object | New entry required |
| | objname | Set same as objname in ufos_field. |
| | numfield | Set = max[fldno] (1 in this case). |
| | objtype | Set = "SUBSCRIPT". |

You can re-check that the UFD file creates a valid UDD at this stage, however there are some simple changes that are worth making to other tables:

| cobol_field | ufos_field | Changes to existing entry |
|--------------------|------------------------|--|
| tablename | tablename | The existing name of the table. |
| arrays | ----- | Set to 1 as array processing is required. |
| ----- | nobj | Increase by 1 for additional entry in 'ufos_tab_object'. |
| recexpr | cond | Used later if necessary to limit rows returned. |
| | ufos_tab_object | New entry required |
| | tablename | The name of the table. |
| | objno | Current highest object number for this tablename + 1. |
| | object | The name of the "SUBSCRIPT" object (same as objname in ufos_field) Set = "SUB" in this example. |

Note: As in [step 1](#), it is recommended that you check the UFD text file entries you have created will build into a valid and working UDD before proceeding to the next step.

Notice that the extra column, **IDX** in this example, is visible alongside the other columns in the table, although the contents are, at this stage, incorrect

Step 4

In order to get the subscript field to cycle through the required range of 1 to 'n', it is necessary to add an entry into each of the array system tables.

The contents of these tables are listed below:

| array_object | ufos_array_object | New entry required |
|---------------------|--------------------------|--|
| ----- | name | An unique name used to identify the array. Same as the 'objname' in the 'array_level' table. For example, "ARRY". |
| objid | ----- | An unique number, in this table, to cross-reference to 'objid' in the 'array_level' table. |
| max_levelno | nlevel | The maximum number of subscripts needed to address the array elements (set = 1 in this example). |
| objseq | ----- | Sequence number of entries in this table. |
| array_level | ufos_array_level | New entry required |
| tablename | tablename | Must match the table 'name' used above. |
| levname | objname | An unique 'name' used to identify the array. say (for example) = "ARRY". |
| objid | ----- | Cross-reference to 'objid' in 'array_object' table. |
| levelno | levelno | The 'level' of the subscript to 'cycle' - in this case we only have one, so set = 1. |
| start_offset | start_offset | The offset of the first byte of the first element of the array. |
| element_size | element_size | The size, in bytes, of one element (which may be several fields) of the array. |
| max_elements | max_elements | The maximum number of elements (this is the maximum value of the subscript). |
| min_elements | ----- | The lowest element (usually = 1). |
| depending_on | ----- | * usually set to underscore "_" . |

| | | |
|--------------|-------|--|
| parallel_to | ----- | * usually set to underscore "_". |
| subscript | ----- | Set to the 'fldname' in 'cobol_field' of the subscript field. In this example = "IDX". |
| type | ----- | Set to 0 (zero). |
| is_displayed | ----- | Set to 1. |

The entries above identified with * are described in detail later in this section.

Note: As in [step 1](#), it is recommended that you check the UFD text file entries you have created will build into a valid and working UDD before proceeding to the next step.

The subscript, **IDX** in this example, now cycles through the range 1 to n, and the number of rows returned is increased by the factor of 'n'.

Step 5

The final stage is to 'tidy up' the data retrieved from the array.

If you are using the **cobol_field** model, it is necessary to amend the entries in the **cobol_field** table for the field(s) that make up the array element as follows:

Bit 5 of the **fieldflags** column must be set. This is equivalent to adding 16 to the existing value.

Enter the 'levname' used in the controlling 'array_level' table in the **arraylevname** column; ARRY in this example.

Enter the 'levelno' used for the appropriate subscript in the controlling 'array_level' table in the **arraylevel** column; 1 in this example.

It may be necessary to consider ignoring some of the data. For example if not all the elements of the array contain data then they should not become a row in the **repeating group** table as a

```
select count(*) from <table>;
```

will include these rows in the count. This is achieved by inserting a suitable expression in the **recexpr** or **cond** column (the name of this column depends on your DSD) of the **xxx_table** entry, from [step 3](#) above, to filter out any unwanted rows.

The entries for the repeating group table are now complete.

Note: As in [step 1](#), it is recommended that you check the UFD text file entries you have created will build into a valid and working UDD before including these entries into a 'production' UDD.

Special Considerations

Parallel OCCURS

This special case exists when there are two or more data arrays in a record and the elements of each array correspond, that is the second element of array 1 corresponds to second element of array 2 and so on.

Note: *Where there are several arrays in the record and these contain differing numbers of elements, then the record must be mapped onto more than one table, which can then be joined by the SQL query when necessary.*

The only Data Source Drivers that enable support for parallel occurs to be mapped onto a single table are those that have a **parallel_to** column in the **array_level** table, see [step 4](#).

The process of setting up the UFD text file is exactly the same as described in steps 1 to 5 of the [Repeating Groups](#) section, except that a separate entry is required in the **array_object** table for each array that is participating in the data table.

The differences between the entries are highlighted below.

| | |
|---------------------|---|
| array_object | New entry required |
| objid | An unique number, in this table, to cross-reference to 'objid' in the 'array_level' table. |
| max_levelno | The maximum number of subscripts needed to address the array elements (set = 1 in this example). |
| objseq | Sequence number of entries in this table. |
| array_level | New entry required |
| tablename | Must match the 'table name' used above. |
| Levname | An unique 'name' used to identify the array. For example, say, "ARRAY2". |
| objid | Cross-reference to 'objid' in 'array_object'. Set to number of previous objid + 1 for second array. |
| levelno | The 'level' of the subscript to 'cycle' - in this case we only have one, so set to '1'. |
| start_offset | The offset of the first byte of the first element of the required array. |
| element_size | The size, in bytes, of one element of the array, which may be several fields. |
| max_elements | The maximum number of elements (this is the maximum value of the subscript). |
| min_elements | The lowest element (usually = 1). |
| depending_on | * usually set to underscore "_". |

| | |
|--------------|---|
| parallel_to | Set to the "levname" of the first array (= "ARRY" for example). |
| subscript | Set to the 'fldname' in 'cobol_field' of the subscript field (In this example = "IDX"). |
| type | Set to 0 (zero). |
| is_displayed | Set to 1. |

This mechanism enables entries in two or more arrays to be processed as a parallel OCCURS type of repeating group mapping.

Nested OCCURS

This special case exists when the data array requires two or more subscripts to address the element concerned, that is it is a multi-dimension array.

Consider the following example:

```

01          SALEHIST.
           03          STKNUM          PIC X(6) .
           03          NO-OF-ITEMS    PIC 999.
           03          STK-DATA OCCURS 2.
                05     CUSTCODE OCCURS 5.
                05     YEAR          PIC 99.
                05     SALES         PIC 9(5)V99.
           03          TOTAL-SALES    PIC 9(5)V99.
    
```

The process of setting up the UFD text file is exactly the same as described in steps 1 to 5, above, except that two subscript entries are required in the **xxx-field** table, say, **IDX1** and **IDX2**, and a separate entry is required in the **array_level** table for each array subscript needed to address the fields of the elements participating in the data table.

The following table shows the additional entries required to define the two subscript entries needed.

| cobol_field | ufos_field | Modified (2) entries required. (other fields as in earlier example) |
|-------------|------------|--|
| ----- | objname | Cross reference name, say SUB, to entry in 'ufos_object' of 'objtype' = 'SUBSCRIPT'. |
| ----- | fldno | Identify field within object (1 for first subscript 2 for second subscript). |
| tabname | ----- | Table name. |
| fldname | fldname | An unique identifier (use IDX1 & IDX2 for this |

| | | |
|--------------|--------------------|---|
| | | example). |
| arraylevname | ----- | An unique identifier (say ARRO & ARRI for outer and inner array). |
| arraylevel | ----- | 'Depth' of array (1 for outer array, 2 for inner array entry). |
| | ufos_object | Modified single entry |
| | objname | Set same as 'objname' in 'ufos_field'. |
| | numfield | Set = max[fldno] (2 in this case). |
| | objtype | Set = "SUBSCRIPT". |

It is also necessary to add additional data to both array tables to process the additional subscript level that is necessary.

| array_object | ufos_array_object | Modified entry required |
|---------------------|--------------------------|---|
| ----- | name | An unique 'name' used to identify the array. Same as the 'objname' in the 'array_level' table. For example, say = "ARRY". |
| objid | ----- | An unique number, in this table, to cross-reference to 'objid' in the 'array_level' table. |
| max_levelno | nlevel | The maximum number of subscripts needed to address the array elements (set = 2 in this example). |
| objseq | ----- | Sequence number of entries in this table. |
| array_level | ufos_array_level | Two entries required |
| tablename | tablename | Must match the 'table name' used above. |
| levname | objname | An unique 'name' used to identify the array. For example, say, = "ARRY". |
| objid | ----- | Cross-reference to 'objid' in 'array_object'. |
| levelno | levelno | The 'level' of the subscript to 'cycle' - in this case we have two entries, so set first entry to '1' second entry to '2'. |
| start_offset | start_offset | The offset of the first byte of the first element of the array. |
| element_size | element_size | The size (in bytes) of the element of the array addressed. Note, the level 1 entry addresses an element of 45, the level 2 entry an element of size 9 (in example). |

| | | |
|--------------|--------------|---|
| max_elements | max_elements | The maximum number of elements (this is the maximum value of each subscript - 2 for level 1 entry 5 for level 2 entry). |
| min_elements | ----- | The lowest element (usually = 1). |
| depending_on | ----- | * usually set to underscore "_". |
| parallel_to | ----- | * usually set to underscore "_". |
| subscript | ----- | Set to the "fldname" in 'cobol_field' of the subscript field (In this example say level 1 entry = "IDX1" and level 2 entry = "IDX2"). |
| type | ----- | Set to 0 (zero). |
| is_displayed | ----- | Set to 1. |

This mechanism enables entries in a two or more dimensional array to be processed as a **nested OCCURS** type of **repeating group** mapping.

OCCURS Depending on

This special case exists when the number of elements in the data array is dependent on another field in the record.

Consider the following example:

```

01          SALEHIST.

          03          STKNUM                      PIC X(6) .

          03          NO-OF-ITEMS                 PIC 999.

          03          STK-DATA OCCURS 0 TO 100
                    TIMES
                    DEPENDING ON NO-OF-ITEMS.

          05          CUSTCODE                     PIC X(4) .

          05          YEAR                        PIC 99.

          05          SALES                       PIC 9(5)V99.

          03          TOTAL-SALES                 PIC 9(5)V99.
    
```

Where the **array_level** table has a **depending_on** column, then this must be set to the **fldname** of the controlling field; "NO-OF-ITEMS" in this example.

| array_level | Modified entry required |
|--------------|--|
| depending_on | Set to value of fldname entry in cobol_field table that controls this array. |

Where this feature is not available, an entry in the **recexpr** or **cond** column of the **xxx_table** will filter out the unwanted rows.

| cobol_table | ufos_table | Change to existing entry |
|-------------|------------|---|
| recexpr | cond | In this example, if subscript = "IDX", then idx <= no_of_items will filter the unwanted rows. |

This mechanism enables entries in an array holding variable numbers of elements to be processed as a **depending on** type of **repeating group** mapping.

Note: *To handle data arrays efficiently, it may be necessary to consider the best way of mapping the data array onto a 'relational' style table. It is generally best to experiment in a test environment before modifying the production version of the UDD.*

Limitations

- The syntax for UNION ALL is supported. However, if the tables hold different numbers of columns an error is generated. If the formats are different they are converted to a common format, usually CHAR type.
- Rev 3.10 and all previous versions of U/SQL, use floating point arithmetic for internal computation of integer values. This may result in rounding errors of the least significant digit, which is evident when the number is displayed with 16 or more significant digits.
- When using the Character Translation table, it is not possible to use scalar functions within a record expression as there is a conflict between the needs of foreign character translation and the ODBC syntax for scalar functions, which prevent them being stored in a database.
- 'Grant and Revoke' is a sub-set of the ANSI standard [Grant/Revoke security system](#), providing that the dba user is the only user with GRANT permission, and may not pass such permission to another user.
- 'Select for update' has been implemented with a mechanism that locks one row at a time and not the entire 'cursor'. This is only effective when using an update or delete 'where current of <cursorname>'.

Note: *This does not apply if the [TrueSFU](#) directive has been set.*

- When importing or exporting tables using [Win U/SQLi](#), the layout of data can sometimes leave the text with line lengths that are longer than that supported by standard editors.
- The U/SQL Manager does not support ACUCOBOL **XFD** directives.
- Dates and times are validated when converted to/from physical storage. This ensures that invalid data cannot be stored, nor can it be retrieved without error (these data items are normally a sign of an invalid database definition in the UDD). If this is not desired, that is, invalid dates are desired, then set the directive **AllowInvalidDates=Y** for that data source.
- There is a limit of 250 DSNs within a single **usqlsd.ini** file or within a single service on Windows NT. To exceed this limit use the **MaxDSN** directive.
- U/SQL 3.20 does only allow you to access U/FOS datasources if you are using U/FOS NT 2.4.2042 and above. If you attempt to access a U/FOS datasource using U/SQL 3.20 against an older version of U/FOS NT (for example, 2.4.1017 / 2039), the following error is generated:

```
Procedure entry point db_term could not be located in the dynamic
link library libkernel.dll
```

Specific UDD Information

Setting Up a COBOL Data Dictionary

Planning to Use U/SQL Manager

Note: Refer to the [Overview](#) and [Creating a UDD](#) sections for an overview of the dictionary technology and the steps involved in creating a UDD.

For ACUCOBOL Vision and Micro Focus EXTFH files, creating the UDD has been made as easy as possible. The U/SQL Manager helps you build the dictionary using the information declared by your COBOL FDs, **SELECT** statements and appropriate copyfiles.

Note: The U/SQL Manager is only available as a 32-bit client product.

This section:

- Summarizes the steps required to set up a dictionary using the U/SQL Manager.
- Describes what you need to do before you start by providing an overview of the way COBOL data is represented, and the decisions you may have to make when planning your COBOL data dictionary.
- Describes expression handling and its syntax. For example, to differentiate between multiple record types within the same physical file, as separate logical tables, you are able to specify an expression that uniquely distinguishes each record type from the others.

U/SQL Manager Revision Control

The U/SQL Manager checks both the U/SQL Server engine (Single or Multiple-tier) and dictionary it is connected to. It will not allow connections to engines it deems are not compatible, and will warn, but still allow a connection to a UDD that it deems to be incompatible.

To obtain the minimum versions of both the engine and the dictionary expected by the U/SQL Manager display its **About** box (located in the **Help** menu).

Summary of the Steps to Set Up a Dictionary

1. Read the documentation, in particular the section Representing COBOL Data below, and plan what you intend to achieve.
2. Copy, into a suitable directory on your PC, one or more COBOL programs or copyfiles containing the appropriate COBOL FDs and SELECT statements.
3. Start the U/SQL Server on your host system (Multiple-tier only).
4. Start the U/SQL Manager on your PC.
5. Either create a new data dictionary (or open an existing one if you want to modify or add to it).
6. Select the first COBOL program or copyfile you wish to use and the COBOL FD Converter will parse it to obtain the record names.
7. If necessary, rename the records (tables) to something more suitable.
8. Select the first record (table) to be further parsed for File, Data and index Key information.

9. Manipulate the Data:

- If necessary, modify the field (column) names to be compliant with SQL syntax. Specify with dates and NULL fields.
- Expressions can be added to columns or virtual columns to change their values.
- Include or exclude REDEFINES, and, if required, exclude other fields.
- If you have multiple 01 records, multiple logical tables will be created for the same file. Remember you need to supply an expression that distinguishes each different record type.
- Consider OCCURS statements and the possible combinations of creating individual fields, repeating groups, parallel OCCURS, nested OCCURS and OCCURS DEPENDING ON.

10. Check that all the index Key information has been created successfully from the SELECT statements, if they are available. Otherwise, you must enter all the index key information manually.

11. Each logical table has to be given its physical file name. Check that the organization of the file, for instance, INDEXED, is correct.

If there is more than one logical table for the same physical file, then you need to enter the expression that differentiates the record type of each logical table.

12. When satisfied, use the U/SQL Manager to write the information to the dictionary. Note that the process of loading your data into the dictionary automatically creates the logical or table views that will be visible to the ODBC-enabled products.

13. If you do not have a COBOL FD, you can enter the data, index and file information manually.

14. Repeat the above procedure for all your application's records and files until the UDD is complete.

The remainder of this section discusses in more detail how your COBOL data may be represented and the decisions you may have to make when planning your COBOL data dictionary.

Refer to the sections [Using U/SQL Manager to Create a UDD](#) and [Modifying a UDD](#).

Representing COBOL Data

Representing COBOL Data

The following sections provide an overview of the way COBOL data is represented, and the decisions you may have to make when planning your COBOL data dictionary.

- [File Access Methods Supported](#)
- [Dealing with Multiple 01 Level Records](#)
- [Expression Handling Syntax](#)
- [Field \(Column\) Names](#)
- [Dates](#)
- [Defining NULL Columns \(Fields\)](#)
- [Expressions on Columns and Virtual Columns](#)
- [REDEFINES](#)
- [OCCURS](#).

It is assumed that you are familiar with at least the following:

- The COBOL file organization types (indexed, relative and so on) used in your data files.
- The compiler and storage options used to create your data files. A number of things can be affected by these settings so it is important that the correct ones are known and understood. For example, the record length can be adjusted dependent on the settings.
- How REDEFINES have been used in your data and, if separate logical tables apply to each re-definition, what value or expression signifies the record type for each redefine.
- For multiple 01 level record types, what they mean in your application and what value or expression signifies each record type.
- If variable length records are used you should be aware of the meaning and impact of NULL of columns in the variable portion of the record. These records are restricted to read only.
- Ensure record FDs contain all their fields including any embedded or trailing FILLER. If they do not then incorrect record lengths may be calculated.
- The **SELECT** statements containing the index key components of your data. If your COBOL FDs qualify data names in the specification of any index, then the U/SQL Manager may not correctly identify these key field.

It is extremely important to check that the entries for each index contain all the key components in the correct order. Failure to do so will result in performance degradation as the index will not be used in certain cases, when it should be, or even in U/SQL not being able to perform the query at all.

Check the contents of your indexes using the appropriate COBOL utility:

- ACUCOBOL - utility **vutil**
- Micro Focus COBOL - utility **fhinfo**.

Refer to your COBOL documentation for further details.

- Your application's data to ensure that you are obtaining the correct results once you have built your UDD and started to use U/SQL with an ODBC-enabled product.

File Access Methods Supported

For each file, you will have to consider its access method, what indexes it has and how it will be used. U/SQL automatically recognizes and supports the following file types:

- Indexed (fixed and variable record length)
- Relative (fixed and variable record length)
- Sequential (fixed and variable record length)
- Line Sequential (terminated by either newline or carriage return/newline, dependent on operating system).

Dealing with Multiple 01 Level Records

If there are multiple 01 records defined for the same physical data file, each record type is described as a separate logical table. You will need to know how each record type can be distinguished from the others. This is usually determined by one or more fields having particular values. You must decide on the name you wish to use for each logical table and establish the differentiating values or expressions. This is known as a record expression.

When you subsequently access each logical table via an ODBC-enabled product, the U/SQL Server will automatically use the differentiating values or expressions to return only the appropriate records from the physical file.

Note: *If you have multiple records or structures, in the same physical file, you must define each one as a separate logical table.*

The following example FD of an employee file (EMPL) has two 01 records - the Employee Master Record and the Employee Salary Record.

```
FD EMPL.
```

*** Employee Master Record**

```
01 MAS-EMPREC.
  03 MAS-EMP-KEY.
    05 MAS-COMPANY          PIC 9(2).
    05 MAS-NUMBER           PIC 9(5).
    05 MAS-REC-TYPE         PIC X.
* MASTER RECORD IS RECORD TYPE = "M"
  03 MAS-ALT-KEY.
    05 MAS-LASTNAME         PIC X(20).
  03 MAS-INITIALS           PIC X(4).
  03 MAS-SSNUMBER           PIC X(15).
  03 MAS-ADDRESS.
    05 MAS-ADDRESS1         PIC X(15).
    05 MAS-ADDRESS2         PIC X(15).
    05 MAS-ADDRESS3         PIC X(15).
    05 MAS-ZIP-CODE         PIC 9(5).
  03 MAS-PHONE              PIC X(13).
  03 MAS-FROM-DTE           PIC 9(6).
```

*** Employee Salary Record**

```
01 SAL-SALREC.
  03 SAL-SAL-KEY.
    05 SAL-COMPANY          PIC 9(2).
    05 SAL-NUMBER           PIC 9(5).
    05 SAL-REC-TYPE         PIC X.
* SALARY RECORD HAS RECORD TYPE = "S"
  03 SAL-ALT-KEY.
    05 SAL-GRADE            PIC X(20).
  03 SAL-TITLE              PIC X(10).
  03 SAL-SALARY             PIC S9(5)V9(2)
                           TRAILING SEPARATE.
  03 SAL-I-AMT              PIC S9(5)V9(2)
                           TRAILING SEPARATE
                           OCCURS 3 TIMES.
  03 SAL-I-DTE PIC 9(6)     OCCURS 3 TIMES.
```

The U/SQL Manager's COBOL FD Converter will parse this FD and create two logical tables, one called MAS_EMPREC and the other SAL_SALREC. You can rename them to, say, Emp_Master and Emp_Salary.

The two record types are distinguished by the REC_TYPE field being "M" or "S" respectively. Thus for the MAS_EMPREC table you will insert the **record expression**:

```
MAS_REC_TYPE="M"
```

and for the SAL_SALREC table the **record expression** will be:

```
SAL_REC_TYPE="S".
```

Notes:

1. *The field separator character '-' is illegal in SQL syntax. All '-' characters are automatically converted to '_' by the U/SQL Manager's COBOL FD Converter.*
2. *The names of any fields in the Expression are NOT case-sensitive.*
3. *See the [Expression Handling Syntax](#) section.*
4. *The record expression may not include items which are parts of repeating groups.*

Expression Handling Syntax

As described in the [Dealing with Multiple 01 Level Records](#) section, multiple 01 level records can be denoted as separate logical tables distinguished by including a differentiating value or expression with each record definition in the UDD.

U/SQL includes an expression handler that conforms to the X/Open SQL syntax for the **WHERE** clause search-condition that qualifies the selection of query rows.

Search conditions are one or more predicates, combined with the logical operators **AND**, **OR** and **NOT**.

The names of columns (fields) in expressions are not case-sensitive.

Note: *Scalar functions and sub-queries within the search-condition are not supported.*

The following sections provide the search-condition syntax.

Logical Operators

The order of precedence among the logical operators is **NOT**, followed by **AND**, followed by **OR**. The order of evaluation at the same precedence level is from left to right. Parentheses can be used to change this order.

Comparison Predicate

A comparison predicate compares two values and has the form:

expression_1 comparison_operator expression_2

where *comparison_operator* can be any of the following:

- = equal to
- <> not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

The result is true or false, depending on the outcome of the comparison.

BETWEEN Predicate

A BETWEEN predicate tests whether a value is within a range of values and has the form:

expression_1 [NOT] BETWEEN *expression_2* AND *expression_3*

The predicate (without **NOT**) is equivalent to:

expression_1 >= *expression_2* AND *expression_1* <= *expression_3*

Using the keyword **NOT** negates the result in the manner of the **NOT** logical operator.

LIKE Predicate

A LIKE predicate compares a column value with a pattern and has the form:

```
[NOT] LIKE pattern_value
```

The column `column_name` must reference a character string column, and `pattern_value` is a **character string literal**.

The result of the predicate is true or false depending on whether or not the value of the column referenced by `column_name` conforms to the specified pattern. Using the keyword **NOT** negates the result in the manner of the **NOT** logical operator.

Pattern Syntax

Within the **character string literal** represented by `pattern_value` in the above example, characters are interpreted as follows:

- The underscore character '_' stands for any single character.
- The percent character '%' stands for any sequence of zero or more characters.
- All other characters stand for themselves.

For example:

- `LIKE '%X%'` is true for any column value that contains the character X.
- `LIKE 'Y_'` is true for any column that is two characters wide and starts with the character Y.

Note: The **character string literal** can be bounded by either single quotes (') or double quotes (").

Field (Column) Names

Field (column) names must be unique within a table, normally up to 18 characters conforming to SQL naming standards, but can optionally be a maximum of 30 characters. The U/SQL Manager's COBOL FD Converter checks that all field names do comply and notifies you if any do not. Compliant field names can be generated automatically, if required.

Dates

U/SQL is capable of manipulating a broad range of physical date formats which it maps to the ODBC SQL Date data type (CCYYMMDD) that then allows date formatting and sorting by ODBC-enabled products.

You need to know which fields are to be declared as dates and whether they are defined as a format string, say, DD/MM/YY, or as a Julian date, in which case you will need to know the base or epoch date, for instance, 01/01/1980.

Defining NULL Columns (Fields)

It is often useful if a particular value in a column (field) can be considered the SQL **NULL** value.

NULL values are determined by a boolean expression in the form of a **search_condition** placed on any column which, if true, indicates that the column is NULL. The **search_condition** is defined in the [Expression Handling Syntax](#) section.

You define the **NULL** value using any of the following formats:

- "X" - where X is a printable ASCII character.
- xHH - where HH is a hexadecimal representation of an ASCII character.
- DDD - where DDD is a decimal representation of an ASCII character.

If the outgoing data is to be represented in EBCDIC, it is converted from ASCII to EBCDIC after NULL value conversion takes place.

For example, in an Employee table (record) there is a column (field) LEFT_DATE which, if non-zero, indicates the date on which an employee left the company. This column could have the NULL boolean expression as:

0 : zero

to indicate a NULL date.

The following SQL query could then be used to select all past employees:

```
select * from employee where left_date is not null;
```

Note: *U/SQL implies NULL in certain situations, for example, if the sign 'nibble' (half byte) of a COMP-3 is invalid.*

Expressions on Columns and Virtual Columns

An expression can be placed on any column (field) to change its value, depending on the value of other columns in the same row (record).

The column expression has three forms:

1. `IF search_condition THEN assignment_expression [ELSE assignment_expression]`
2. `assignment_expression`
3. `$special_assignment_expression`

where `search_condition` and `assignment_expression` are defined in the [Expression Handling Syntax](#) section.

The expression forms applicable to columns and virtual columns are summarized in the following table:

| Column | Virtual Column |
|--|---|
| Expression form 2. Note: Such a column cannot be updated. | Expression forms 1 and 2 |
| Expression form 3. Note: Such a column can be updated. | Note: Virtual columns cannot be updated. Update the real columns only. |

Note: Column expression and Date format are mutually exclusive in any column definition.

`assignment_expression` can be any of the following:

- a column name, for example, `AMOUNT`
- a string literal or numeric constant, for example, `"Y"` or `10`
- an arbitrary arithmetic expression, for example:

```
( (FLAG2-3) / 6) - (AMOUNT * 10) + FLAG2
```

For example, assume a data record has the column `BALANCE` which always holds the amount as a positive value. A further column `CREDIT` is set to `"Y"` if the `BALANCE` amount is negative. For an ODBC-enabled product to be able to easily provide the summation of the amounts, taking account of whether they are debit or credit values, from a number of rows (records), the value in `BALANCE` needs to be modified.

This is achieved by placing an expression on the column `BALANCE` to modify its value, after it has been read, to be negative if the column `CREDIT` is set to `"Y"`. The column expression could be either:

```
IF CREDIT="Y" THEN -BALANCE
```

or

```
IF CREDIT<>'Y' THEN BALANCE ELSE -BALANCE
```

\$special_assignment_expression

The internal UFD table, **cobol_field**, of the COBOL data source driver has had the column **colassignexpr** enhanced. Refer to section Step 2: Define Fields (Columns) in Chapter 5, Manually Create a COBOL UDD.

The use of this field has been extended to allow syntax **\$nnnn** where:

- '\$' has been introduced as a special directive to indicate the current field addressed by the entry, and
- 'nnnn' is an expression that resolves to a numeric constant.

For example, suppose the field is a record_type and is stored with values 18, 19 and 20 but the end user knows these as 1, 2 and 3 (say), then the 'perceived' value of the column must be adjusted by subtracting 17.

This is achieved by setting the value in **column expression (colassignexpr)** to **\$17**.

Note: *The value 17 is positive - it is the value that manipulates the 'perceived' value to make the 'stored' value, not the other way around. An expression of \$-17 would manipulate the column to be perceived to contain 35, 36 and 37 in this example.*

This value takes effect before any filter applied to the **record expression (recexpr)** column of the **cobol_table** in the internal UFD.

Given the correct value, the process is reversible - the value is subtracted on a **SELECT** and added on an **INSERT** or **UPDATE**.

There are limitations:

- The fields must be numeric in nature - they cannot be alpha-numeric.
- The field cannot be redefined and displayed as part of a group field. To do so, the raw value is processed.

Note: *Scalar functions cannot be used in the columns **record expression (recexpr)** or **column expression (colassignexpr)**.*

Virtual Column

A **virtual column** can be defined in any table, where the value is determined solely from a **column expression**. A new column can only be added to the table after the original record's FD details have been entered into the dictionary. In the example above, a new virtual column, say, REAL_VALUE, would then need to have the column expression:

```
IF CREDIT<>'Y' THEN BALANCE ELSE -BALANCE
```

REDEFINES

For COBOL REDEFINES, the dictionary is able to hold both the field or its re-definitions. This means that you are able to access the main field or the components from ODBC-enabled products. In the following example, you would be able to access all three field items, CODE-FIELD, NAME_PART and NUMBER.

```
03 CODE-FIELD          PIC X(16) .
03 CODE-PARTS REDEFINES CODE-FIELD.
   05 NAME_PART        PIC X(10) .
   05 NUMBER           PIC 9(10) COMP-3.
```

Very often REDEFINES are used to specify separate logical tables for each re-definition. You therefore need to apply a **record expression** to each table to identify the appropriate structure. This is similar to having multiple 01 level record types, see the [Dealing with Multiple 01 Level Records](#) section.

Note: *When inserting data into a record with overlapping data, define only one set of overlapping fields - otherwise any difference between the two representations of the data will cause undefined results.*

OCCURS

Note: A detailed discussion on OCCURS is provided in the [Handling Data Arrays](#) section.

U/SQL can handle COBOL OCCURS in several different ways:

- [OCCURS can be flattened out into individual unique fields](#)

or maintained as any combination of the following:

- [Repeating groups](#)
- [Parallel OCCURS](#)
- [Nested OCCURS](#)
- [OCCURS DEPENDING ON \(and, optionally, for Micro Focus, ODOSLIDE\).](#)

OCCURS Flattened Out into Individual Unique Fields

Consider the following example of an ADDRESS that OCCURS three times:

```
03 ADDRESS          PIC X(30)
   OCCURS 3 TIMES.
```

For this type of OCCURS you want to flatten it out into individual fields, one for each address line. This can be automatically actioned by the U/SQL Manager's COBOL FD Converter to give the equivalent relational table:

```
ADDRESS1          char(30)
ADDRESS2          char(30)
ADDRESS3          char(30)
```

Repeating Groups

Consider the following example FD with the JOBS-GROUP occurring 12 times:

```
01 JOBS-RECORD.
   03 JOBS-KEY.
       05 JOBS-NO          PIC X(6) .
   03 JOBS-NAME          PIC X(30) .
   03 JOBS-DESCRIPTION   PIC X(50) .
   03 JOBS-GROUP        OCCURS 12 TIMES.
       05 JOBS-ACCD       PIC S9(10)V9(2) .
       05 JOBS-PAID       PIC S9(10)V9(2) .
```

In a normalized table, JOBS-GROUP appears as 12 separate rows. So, for each JOBS-RECORD there are 12 rows with the same values for JOBS_NO, JOBS_NAME and JOBS_DESCRIPTION repeated in each of the 12 rows:

| Row | JOBS-NO | JOBS-NAME | JOBS-DESCRIPTION | JOBS-ACCD | JOBS-PAID |
|-----|---------|-----------|--------------------|-----------|-----------|
| 1 | 123456 | Anyjob | Anyjob description | 99.99 | 1.2 |
| 2 | 123456 | Anyjob | Anyjob description | 88.88 | 2.30 |

| | | | | | |
|----|--------|--------|--------------------|-------|-------|
| 3 | 123456 | Anyjob | Anyjob description | 77.77 | 3.40 |
| 4 | 123456 | Anyjob | Anyjob description | 66.66 | 4.50 |
| 5 | 123456 | Anyjob | Anyjob description | 55.55 | 5.60 |
| 6 | 123456 | Anyjob | Anyjob description | 44.44 | 6.70 |
| 7 | 123456 | Anyjob | Anyjob description | 33.33 | 7.80 |
| 8 | 123456 | Anyjob | Anyjob description | 22.22 | 8.90 |
| 9 | 123456 | Anyjob | Anyjob description | 11.11 | 9.10 |
| 10 | 123456 | Anyjob | Anyjob description | 12.34 | 10.11 |
| 11 | 123456 | Anyjob | Anyjob description | 56.78 | 11.12 |
| 12 | 123456 | Anyjob | Anyjob description | 91.01 | 12.13 |

When the U/SQL Manager's COBOL FD Converter parses an FD, it automatically assumes that the OCCURS is to be treated as a repeating group and inserts a table column and a subscript column. The table column is given the name of the OCCURS and a suffix of **_TBL** (for example, JOBS_GROUP_TBL). The subscript column is given the name of the OCCURS and a suffix of **_IDX** (for example, JOBS_GROUP_IDX).

This **subscript field** is effectively a key to the repeating group. In the JOBS_GROUP example, JOBS_GROUP_IDX has a value of 1 through 12, identifying each of the repeating rows.

For this example, the equivalent relational table (JOBS_RECORD) is produced:

```

JOBS_NO           char(6)
JOBS_NAME         char(30)
JOBS_DESCRIPTION  char(50)
JOBS_GROUP_IDX    integer
JOBS_ACCD         numeric(12,2)
JOBS_PAID         numeric(12,2)

```

An alternative to maintaining the OCCURS as a repeating group is to define a separate logical table that contains just the key from the first table (needed to join the tables) and the repeating group. For example, a CUSTOMER record may contain many fields with an OCCURS at the end for the 12 months' actual and budget information:

```

01 CUSTOMER.
   03 CUST-KEY.
       05 CUST-NUMBER           PIC 9(5) .
       03 CUST-INITIALS        PIC X(4) .

```

Transoft U/SQL User Guide

```
03  CUST-ADDRESS.  
    05  CUST-ADDRESS1      PIC X(15).  
    05  CUST-ADDRESS2      PIC X(15).  
    05  CUST-ADDRESS3      PIC X(15).  
    05  CUST-ZIP-CODE      PIC 9(5).  
03  CUST-PHONE            PIC X(13).  
. . .  
03  CUST-AMT-GROUP OCCURS 12 TIMES.  
    05  CUST-ACTUAL-AMT    PIC S9(5)V9(2).  
    05  CUST-BUDGET-AMT    PIC S9(5)V9(2).
```

You could declare this one record, CUSTOMER, as two logical tables, one called CUST_MAIN for the name and address details, up to but not including the OCCURS, and the other called CUST_BALANCES, containing the key and the OCCURS only:

Relational table 1, CUST_MAIN, includes the following:

```
CUST_NUMBER      integer  
CUST_INITIALS    char(4)  
CUST_ADDRESS1    char(15)  
CUST_ADDRESS2    char(15)  
CUST_ADDRESS3    char(15)  
CUST_ZIP_CODE    integer  
CUST_PHONE       char(13)  
. . .
```

Relational table 2, CUST_BALANCES, includes the key and OCCURS details:

```
CUST_NUMBER      integer  
CUST_AMT_GROUP_IDX integer  
CUST_ACTUAL_AMT  numeric(7,2)  
CUST_BUDGET_AMT  numeric(7,2)
```

These two tables can be joined by CUST_NUMBER.

Parallel OCCURS

It is possible to have more than one OCCURS defined as repeating groups within the same logical table, known as parallel OCCURS, but they must all have the same number of occurrences. The U/SQL Manager's COBOL FD Converter automatically inserts a single subscript field, called INDEX1, before the first of the repeating groups. You can change this name to one more meaningful to you.

```
01  JOBS-RECORD.  
    03  JOBS-KEY.  
        05  JOBS-NO          PIC X(6).  
    03  JOBS-GROUP.  
        05  JOBS-ACCD        PIC S9(10)V9(2)  
            SIGN TRAILING SEPARATE.  
        05  JOBS-ACM         PIC S9(10)V9(2)  
            SIGN TRAILING SEPARATE  
            OCCURS 12 TIMES.  
        05  JOBS-PAID        PIC S9(10)V9(2)  
            SIGN TRAILING SEPARATE.  
        05  JOBS-PYM         PIC S9(10)V9(2)  
            SIGN TRAILING SEPARATE
```



```

                                OCCURS 12 TIMES.
05  JOBS-BILL                    PIC S9(10)V9(2)
                                SIGN TRAILING SEPARATE.
05  JOBS-BLM                     PIC S9(10)V9(2)
                                SIGN TRAILING SEPARATE
                                OCCURS 12 TIMES.

```

For example, the JOBS-RECORD, above, has three repeating groups each of 12 occurrences. The **subscript field** appears before JOBS-ACM and is automatically created as INDEX1.

This **subscript field** is effectively an extra index key which, in this case, has a value of 1 through 12 for each occurrence row for each JOBS-RECORD.

For this example, the equivalent relational table (JOBS_RECORD) is produced:

```

JOBS_NO           char(6)
JOBS_ACCD         numeric(12,2)
INDEX1           integer
JOBS_ACM          numeric(12,2)
JOBS_PAID         numeric(12,2)
JOBS_PYM          numeric(12,2)
JOBS_BILL         numeric(12,2)
JOBS_BLM          numeric(12,2)

```

You can 'mix & match' the differing types of OCCURS. In this example, each OCCURS could be a separate logical table. Alternatively, one or more of them could be flattened into individual unique fields within the same overall table, or there could be a combination of the two.

Nested OCCURS

COBOL allows multiple nesting of OCCURS. U/SQL supports up to 10 levels of nesting. At each level of nesting, the U/SQL Manager automatically inserts a subscript column. (This is the name of the OCCURS with a suffix of **_IDX**, although you can change the name to be more meaningful to you.)

The **subscript column** is effectively a key to each level of nesting.

In the example, below, two **subscript fields**, JBS_ACCD_IDX and JBS_ACM_IDX, are created.

```

01  JBS-RECORD.
    03  JBS-KEY.
        05  JBS-NO           PIC X(6) .
    03  JBS-ACCD            PIC S9(10)V9(2) OCCURS 12 TIMES.
        05  JBS-ACM          PIC S9(10)V9(2) OCCURS 6 TIMES.

```

In a normalized table, each row includes all the preceding information. So, for each JBS-RECORD, there will be 72 (12 x 6) rows with the same value for JBS-NO.

For this example, the equivalent relational table (JBS_RECORD) is produced:

```

JBS_NO           char(6)
JBS_ACCD_IDX     integer
JBS_ACM_IDX      integer
JOBS_ACM         numeric(12,2)

```

OCCURS DEPENDING ON

COBOL supports the concept of an OCCURS that, for each record, has a variable number of occurrences. This is determined by the OCCURS DEPENDING ON clause. Consider the following example:

```
01  CUSTREC.
    03  CUST-KEY.
        05  CUST-ID                PIC 9(6) .
    03  CUST-NAME                  PIC X(30) .
    03  SALES-COUNT                PIC 9(2) .
    03  SALES OCCURS 1 TO 12 DEPENDING ON SALES-COUNT.
        05  SALES-ACTUAL          PIC S9(10)V9(2) .
        05  SALES-BUDGET          PIC S9(10)V9(2) .
```

This FD shows that the SALES OCCURS can be between 1 and 12 occurrences DEPENDING ON the value of SALES-COUNT.

Note: *The DEPENDING ON field must exist in the record.*

When the U/SQL Manager's COBOL FD Converter parses the FD, it inserts an **OCCURS table field** and a **subscript field**, which have the name of the first field in the OCCURS with **_TBL** and **_IDX** suffixes. You can change these names to ones more meaningful to you.

The U/SQL Manager's COBOL FD Converter registers the fact that this is an OCCURS with a DEPENDING ON field and this is used during the accessing of the table by ODBC-enabled products.

For this example, the equivalent relational table (CUSTREC) is produced:

```
CUST_ID          integer
CUST_NAME        char(30)
SALES_COUNT      smallint
SALES_IDX        integer
SALES_ACTUAL     numeric(12,2)
SALES_BUDGET     numeric(12,2)
```

For Micro Focus COBOL, the compiler directive **ODOSLIDE** is supported. Refer to the section Micro Focus COBOL Compiler Directives in Chapter 3, Using U/SQL Manager to Create a UDD.

Using U/SQL Manager to Create a UDD

Using U/SQL Manager to Create a UDD

This section describes in detail the steps in using the U/SQL Manager to create your dictionary (UDD) for either ACUCOBOL or Micro Focus COBOL data sources:

- [Step 1: Before You Start - Plan](#)
- [Step 2: Copy your FDs to your PC](#)
- [Step 3: Ensure the U/SQL Server is Running \(Multiple-tier only\)](#)
- [Step 4: Start the U/SQL Manager](#)
- [Step 5: Create a New Data Dictionary](#)
- [Step 6: Selecting a COBOL FD File](#)
- [Step 7: Rename Table Names \(optional\)](#)
- [Step 8: Obtain Details of the Table](#)
- [Step 9: Manipulate the Data](#)
- [Step 10: Keys Definition](#)
- [Step 11: File Details](#)
- [Step 12: Write the Table Information to the Data Dictionary](#)
- [Step 13: Create Data, Key and File Information Manually](#)
- [Step 14: Repeat the Procedure for all your Application's Files.](#)

The section [Planning to Use U/SQL Manager](#) summarizes all the steps required to set up a dictionary using the U/SQL Manager. It describes what you need to do before you start by providing an overview of the way COBOL data is represented, and the decisions you may have to make when planning your COBOL data dictionary.

Step 1: Before You Start - Plan

It is important that you plan what you intend to achieve before starting to set up your dictionary. You need to consider the following:

- [Which data files are to be accessed?](#)
- [Availability of your source file\(s\)](#)
- [How your data is to be represented.](#)

Which Data Files are to be Accessed?

You need to decide which of your application data files you want to access via ODBC-enabled products. It is unlikely that you will want all the files used in your application.

Make Sure you have the COBOL FDs

For the selected files, are the COBOL FDs, **SELECT** statements and appropriate copyfiles available to U/SQL? An example **SELECT** statement from the file control section and FD is shown below:

```
SELECT COMPANY
      ASSIGN TO "COMPANY"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS COMPANY-KEY
      LOCK MODE IS AUTOMATIC
      FILE STATUS IS CHK-FILESTAT.
*
FD   COMPANY
      RECORD CONTAINS 32 CHARACTERS.
* Company Record
01  COMPANY-REC.
      03  COMPANY-KEY.
           05  COMPANY-NO           PIC 9(2) .
      03  COMPANY-DESC           PIC X(30) .
```

How U/SQL Adapters Represents COBOL Data

Refer to the [Planning to Use U/SQL Manager](#) section, and the section [Representing COBOL Data](#) that provides an overview of the way COBOL data is represented, and the decisions you may have to make when planning your COBOL data dictionary. The following topics are covered:

- [File Access Methods Supported](#)
- [Dealing with Multiple 01 Records](#)
- [Field Names](#)
- [Dates](#)
- [Defining NULL Columns](#)
- [Expressions on Columns and Virtual Columns](#)
- [COBOL REDEFINES](#)
- [OCCURS.](#)

Once have established what you are trying to achieve, so you are ready to set up your COBOL Universal Data Dictionary.

Step 2: Copy your FDs to your PC

Copy the COBOL FDs-COBOLFDs-->, **SELECT** statements and appropriate copyfiles, from which you want to create your UDD, onto your Windows PC.

The copyfiles can be in different directories to the FDs.

Step 3: Ensure the U/SQL Server is Running (Multiple-tier only)

Before starting the U/SQL Server ensure that you have a client license that allows Read-Write capability and that the U/SQL Server **ReadOnly** directive is not set, otherwise you will not be able to create a UDD.

UNIX

Run the script **check_serv.sh**, in the **bin** directory from the base of the U/SQL Server installation on the UNIX host, to ensure that the Server is running.

```
./check_serv.sh
```

If it is not, run the script **start_serv.sh** in the same **bin** directory.

```
./start_serv.sh
```

Refer to the section [Starting the UNIX U/SQL Server](#).

Windows NT Server

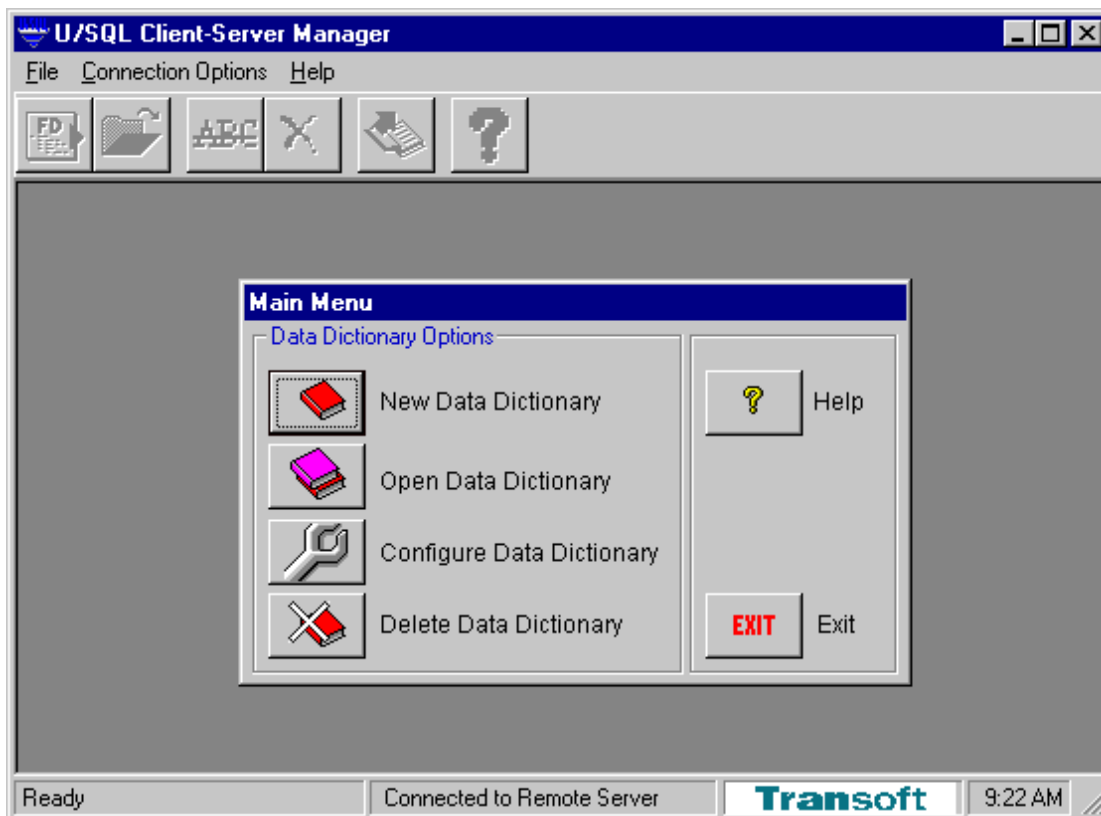
On the Windows NT Server, double-click the **U/SQL Service Manager** icon in the U/SQL program group. Click on the service you wish to start, and select **Manual Start** unless the U/SQL Server service is already running.

Refer to the section [Starting and Stopping the Windows NT U/SQL Server](#).

Step 4: Start the U/SQL Manager

Windows

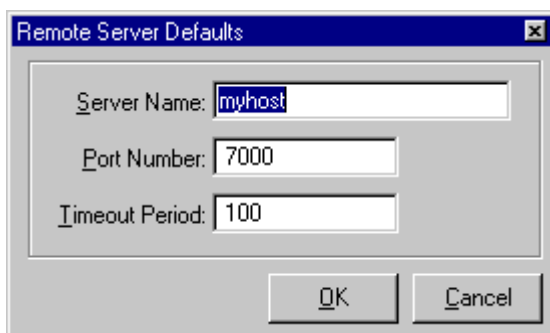
Double-click the **U/SQL Manager** icon in the program group. The **Main Menu** is displayed:



For Multiple-tier installations select the **Remote Server** command from the **Connection Options** menu. For Single-tier installations select the **Local Server** option from the **Connection Options** menu.

The **Connection Options** menu additionally allows you to select the **Remote Server Defaults** and **Local Server Defaults**.

Remote Server Defaults



The **Remote Server Defaults** define the default host Server Name, Port Number and Timeout Period that are used when creating the ODBC entries for a new dictionary in [Step 5: Create a New Data Dictionary](#).

You will have entered the Server Name and TCP/IP Port Number on which the U/SQL Server software were installed when you installed the U/SQL Client software. The Timeout Period defaults to 100 seconds.

The Server Name and Port Number entries are held in the **[Transoft U/SQL Configuration]** section of the client **ODBC.INI** directive settings. For example:

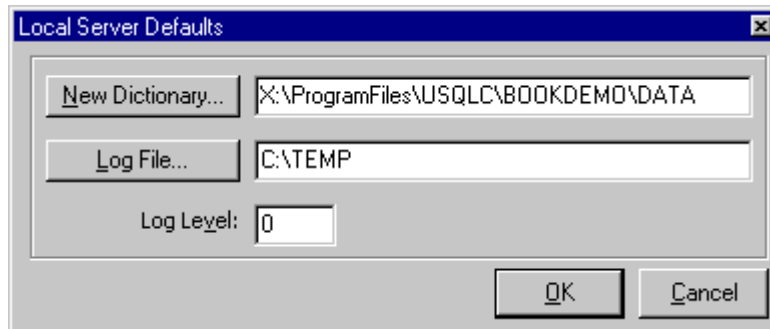
```
[Transoft U/SQL Configuration]
DefaultServer=myhost
DefaultPort=7000
```

`DefaultServer=` entry is overridden by a `Server=` entry within a particular data source's directives. Note, the server name, in this example `myhost`, is case-insensitive.

`DefaultPort=` entry is overridden by a `Port=` entry within a particular data source's directives.

For further details on these and other directives, refer to the section [Client ODBC.INI Directives](#).

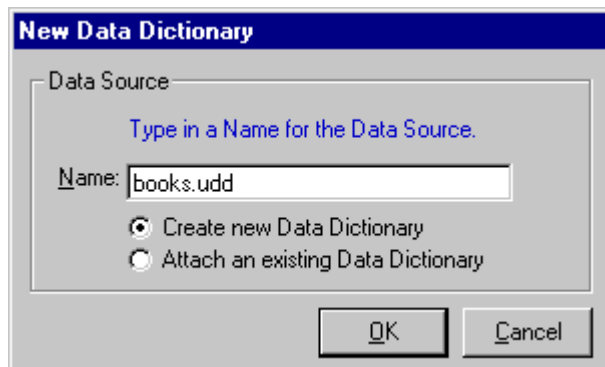
Local Server Defaults



The **Local Server Defaults** define the defaults for the directory where a New Dictionary will be created, the directory where the Log File will be created and the Log Level (this is normally set to zero). Refer to the Single-tier Administration Book for further details on these defaults and log file.

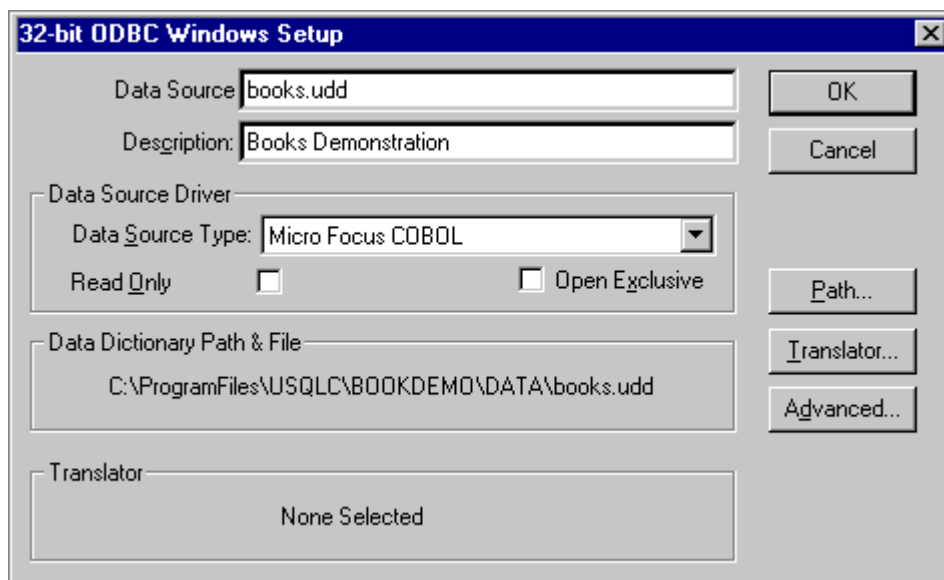
Step 5: Create a New Data Dictionary

Click the **New Data Dictionary** button from the **Main Menu**, or select the **New...** command from the **File** menu. The **New Data Dictionary** dialog box is displayed:



Enter the name you want to give the UDD, for example, books.udd, in the **Name** field. The '.udd' extension is only mandatory for Multiple-tier installations. Ensure the **Create a new Data Dictionary** button is set, then click **OK**. The **32-bit ODBC Windows Setup** dialog box is displayed for either Single or Multiple-tier:

Single-tier



For Single-tier installations, the **Data Source** name is automatically transferred into the first entry box. Notice that the name of the UDD does not have to include the '.udd' extension. You then enter the **Description** that you want to give the data source.

The **Data Source Driver** (DSD) will default to the type for the version of U/SQL you are using, for example Micro Focus COBOL. If you have more than one DSD then they are listed alphabetically in the combo box.

Ensure that both the **Open Exclusive** and **Read Only** check boxes are NOT selected, otherwise you will not be able to create a UDD.

In the **Data Dictionary Path & File** entry box, the path is the default defined by the **Local Server Defaults**, described in [step 4](#), and the dictionary name is the **Data Source** name, with a '.udd' extension automatically added. You can change this by clicking the **Path...** button to display a browser to change the path and UDD name. Note, the '.udd' extension is mandatory.

These are the minimum entries which will be added automatically to the **ODBC.INI** directives when you click **OK** and you are ready to continue with the section Add Data Dictionary Entries.

Additional entries can be entered after clicking the **Translator...** and **Advanced...** buttons. These entries are described in the Single-tier Administration Book.

Multiple-tier

For Multiple-tier installations, the **Data Source Name** is automatically transferred into the first entry box. Notice that the name of the UDD must include the '.udd' extension. You then enter the **Description** that you want to give the data source.

The **Server Name** and **Port Number** will be automatically entered from the equivalent [Remote Server Defaults](#) dialog box. You can change them if you wish to but they must refer to the server name and port number on which the U/SQL Server software is installed.

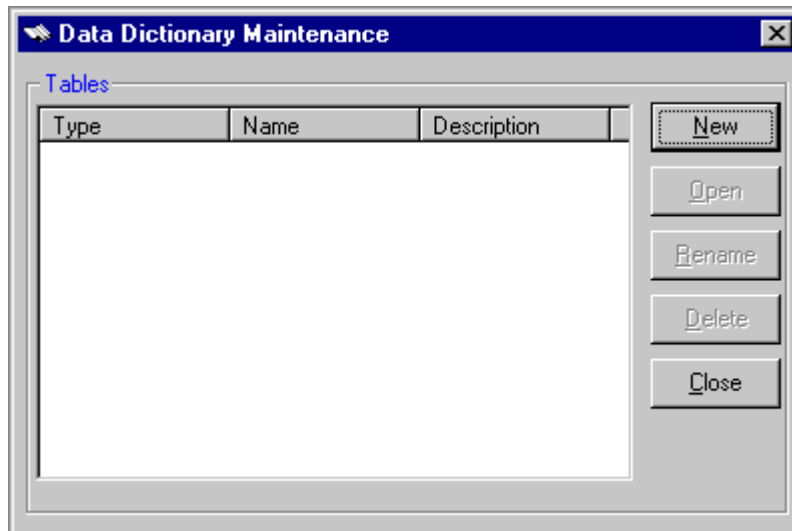
Then enter the **Timeout** value in seconds. The **Timeout** directive represents the time, in seconds, that a client will wait while attempting to connect to the server. The Timeout does not apply after a connection has been established - the client will wait indefinitely for data from a query regardless of this value.

Note: Refer to the section [Client ODBC.INI Directives](#) for further details on these directives.


When you have completed the entries in the dialog box, click **OK** and you are ready to add Data Dictionary entries.

Add Data Dictionary Entries

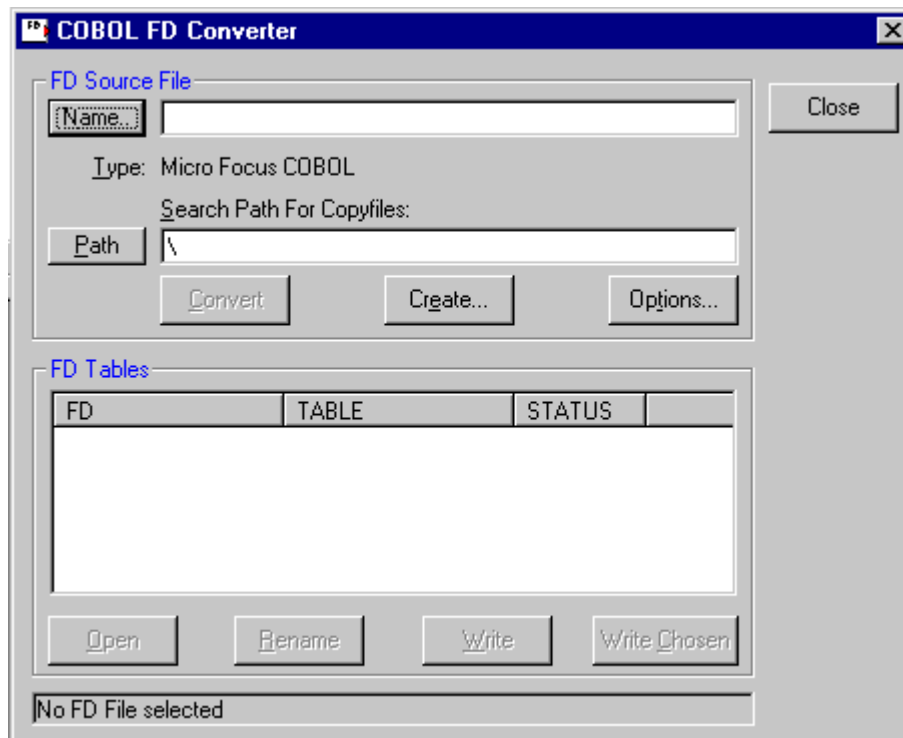
When your empty data dictionary has been created the following form is displayed. If you are opening an existing dictionary, the list of table names that have previously been entered is displayed.



To add entries you can either:

- Click the **FD** button  on the Tool Bar, or
- Click the **New** button on the **Data Dictionary Maintenance** dialog box.

The **COBOL FD Converter** dialog box is displayed:



The Type field confirms whether you are creating a dictionary for either ACUCOBOL or Micro Focus COBOL.

Compiler Options and Directives

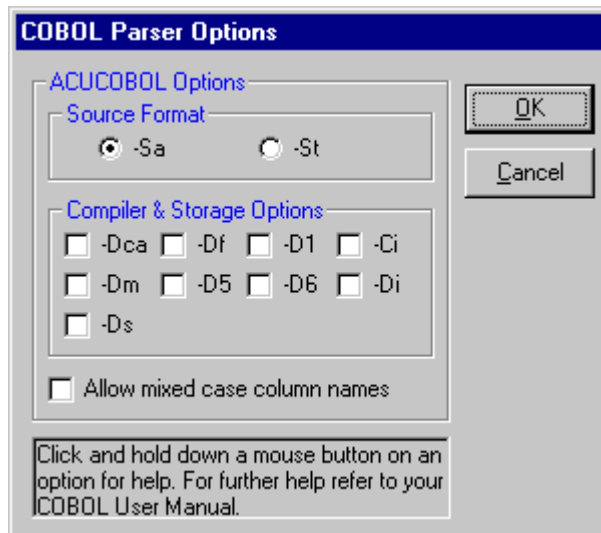
The data stored in your data files is dependent on the compiler and storage options used when the data was created. The record length can be adjusted by the compiler flags.

The compiler can pick up these options and directives from various places, for example, the Micro Focus COBOL **cobopt** file.

Therefore it is important to ensure these options and directives are correctly set in the U/SQL Manager to match those in use.

ACUCOBOL Compiler Options

When you click the **Options...** button, the **COBOL Parser Options** dialog box for the ACUCOBOL Options is displayed:



Set the appropriate ACUCOBOL compiler switches for your data. An explanation of each switch can be obtained by holding down the left mouse button over each option. You will need to set either the **-Sa** or **-St** options, that treat all input source files as ANSI or TERMINAL format, respectively. The U/SQL Manager does not support mixed formats.

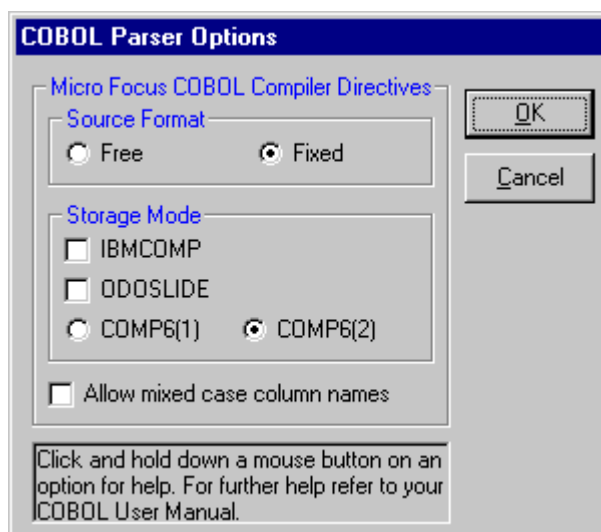
It is very important that you set all other compiler and storage options to those used to create the data files in the first place. For example, if **Dca** is not set properly, the value of 0 is not written correctly to COMP-3 (amongst other things).

Mixed case column names are supported. These are used for report headings. However, column names in SQL statements are case independent.

For further help refer to your ACUCOBOL documentation.

Micro Focus COBOL Compiler Directives

When you select the **Options...** button, the **COBOL Parser Options** dialog box for the Micro Focus COBOL Compiler Directives is displayed:



Set the appropriate source format switch for your FDs. An explanation of each switch can be obtained by holding down the left mouse button over each option.

It is very important that you set all other storage modes to those used to create the data files in the first place. If you do not do so invalid data will be presented to ODBC-enabled products and more importantly if you write back to your data files they will become corrupted.

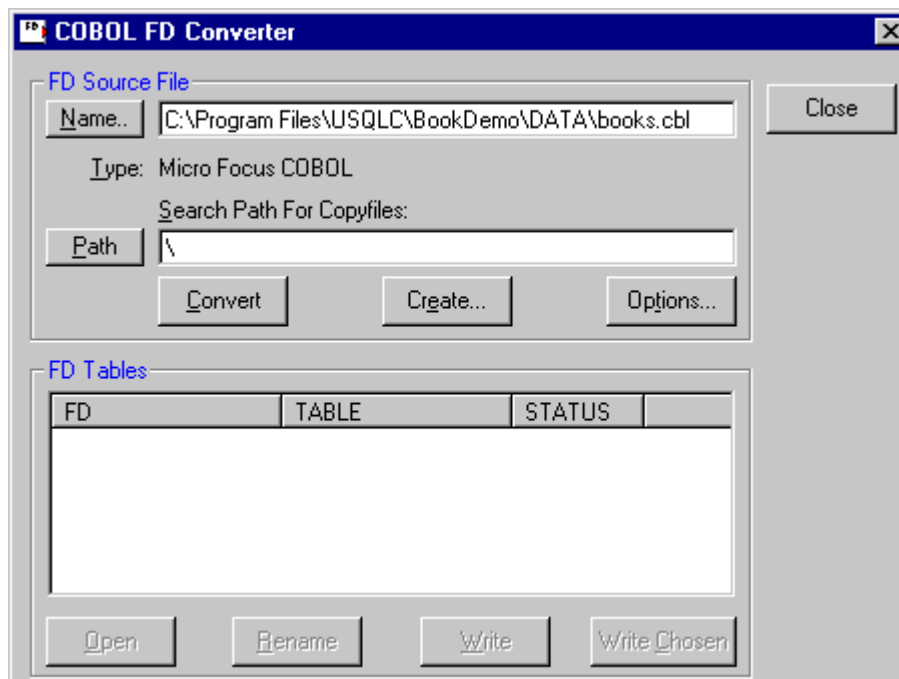
Note, the compiler directive `ODOSLIDE`, indicating `ODOSLIDE` compiler directive (Micro Focus)`OCCURS Depending On Slide` is supported. `ODOSLIDE OCCURSODOSLIDE` (Micro Focus)affects data items that appear after a variable-length array in the same record; that is, after an item with an `OCCURS DEPENDING ON` clause but not subordinate to it. With `ODOSLIDE`, these items always immediately follow the array, whatever its current size; this means their addresses change as the array size changes.

Mixed case column names are supported. These are used for report headings. However, column names in SQL statements are case independent.

For further help refer to your Micro Focus COBOL documentation.

Step 6: Selecting a COBOL FD File

Ensure that the COBOL FD source file(s), from which you wish to create your UDD, are on your Windows PC. Then click the **Name** button on the **COBOL FD Converter** dialog box, which displays the **Select COBOL Source File** browser. Select a COBOL Source File, which can have one of the following extensions '.cbl', '.cpy', '.cpb' or '.cob'. Alternatively, if the source file is not one of these types, use the **All file selection (*.*)**. When you have selected a file, it is displayed in the **COBOL FD Converter** dialog box.



If your selected file requires copyfiles from another directory, click the **Path** button and use the browser to enter their path. Click **Convert** for the COBOL FD Converter to parse this COBOL source file and any associated copyfiles.

The COBOL FD Converter will only parse one source file at a time. However, this file can contain FD details of any number of COBOL files and records.

You do not need to have defined **SELECT** statements. However, if they are not defined, the COBOL FD Converter cannot determine the index key entries and you will have to enter them manually. Alternatively, include them from other files as copyfile entries.

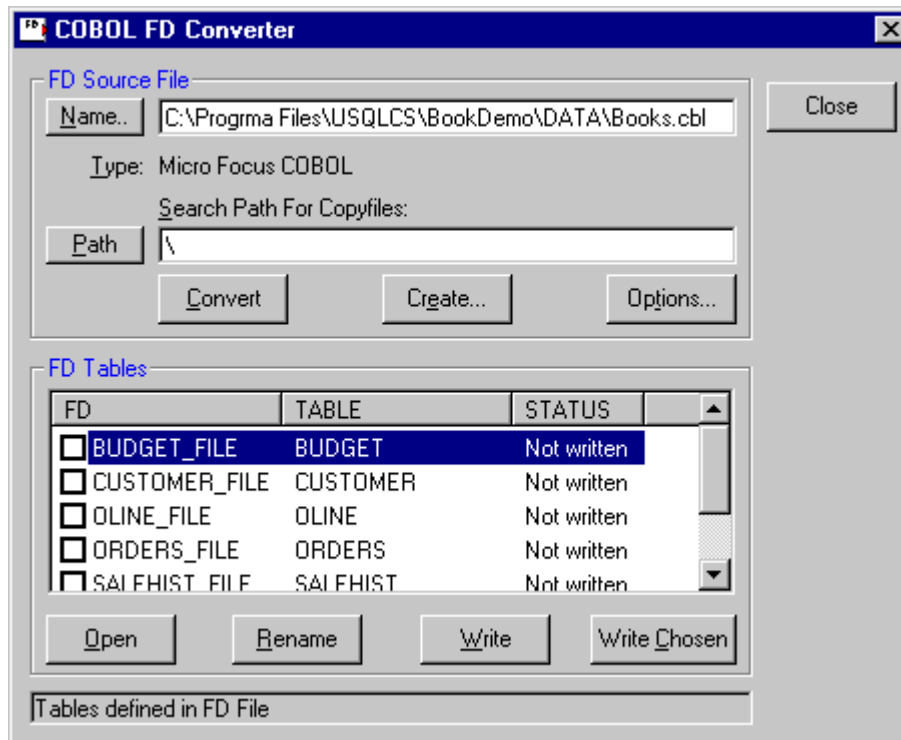
Note: *It is very important to check that each index contains all the key components.*

If no **SELECT** or **FD** statements are found in the file being converted by the COBOL FD Converter, the following message is displayed:

```
FD Converter Error
'C:\Program Files\USQLC\BookDemo\DATA\book.cbl'
```

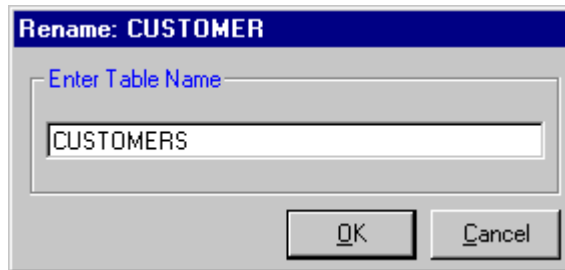
Can't find any SELECT or FD statements in this file. Try changing the Source File Options that are available from the Options button on the FD Converter.

Assuming the COBOL FD Converter has found the relevant information, a list of FD names and Table names is displayed:



Step 7: Rename Table Names (optional)

The **Table** names are visible to the ODBC-enabled products and are the names that are used in all SQL statements. To rename a **Table** name click the **Rename** button. The **Rename: ...** dialog box is displayed:

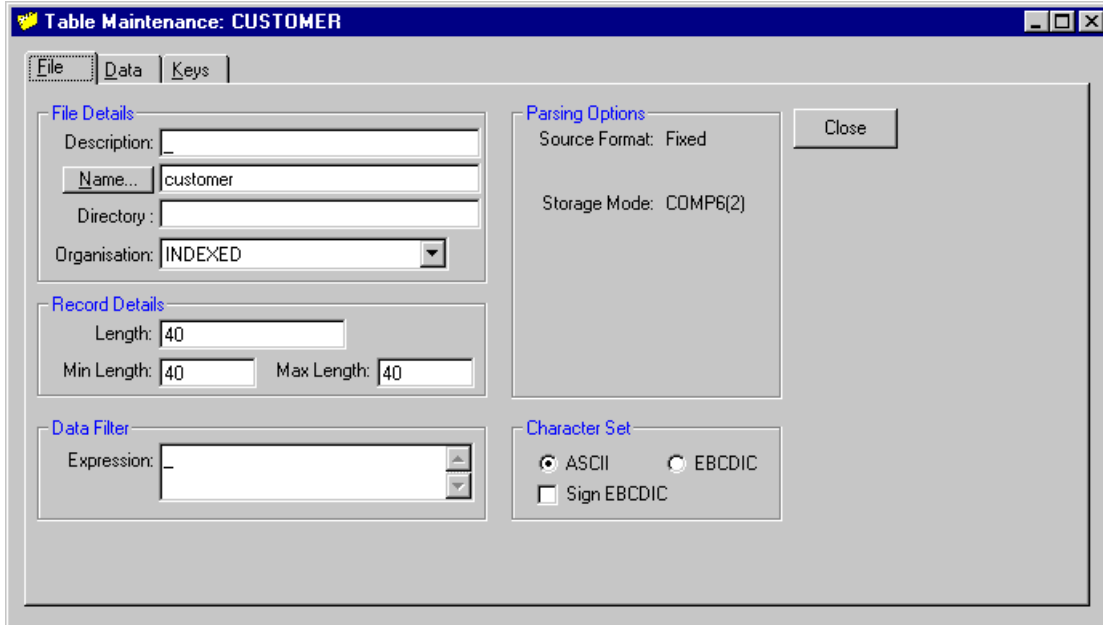


Enter the new name in the **Enter Table Name** field and click **OK**.

Step 8: Obtain Details of the Table

Select an **FD/Table** name, on the **COBOL FD Converter** dialog box, and then either double-click it or click the **Open** button to further parse the FD information.

A property sheet is displayed containing the **File**, **Data** and **Keys** property pages for the selected FD record from the FD information:



Review these property pages in the following order:

- **Data**

This property page contains details of all the fields that make up the FD record. Decide whether you want all these fields to be available to the ODBC-enabled products or a reduced 'view'. **Ensure that all key fields are always included.**

You may want to change the field names to be more meaningful and to comply with SQL syntax. Decide on the treatment of **REDEFINES** - whether you want the main field or its components.

For **dates**, **NULL** fields and **column expressions**, define the format you require.

You must establish how you wish to treat any **OCCURS**.

- **Keys**

If a COBOL **SELECT** statement was available during the parsing of this FD record's details, all the necessary primary and alternate indexes will have been automatically included. Check that they are correct.

If they were not automatically set up, you must enter the details manually. Remember, if you have changed any of the field names in the **Data** property page, these new names must be used in the indexes.

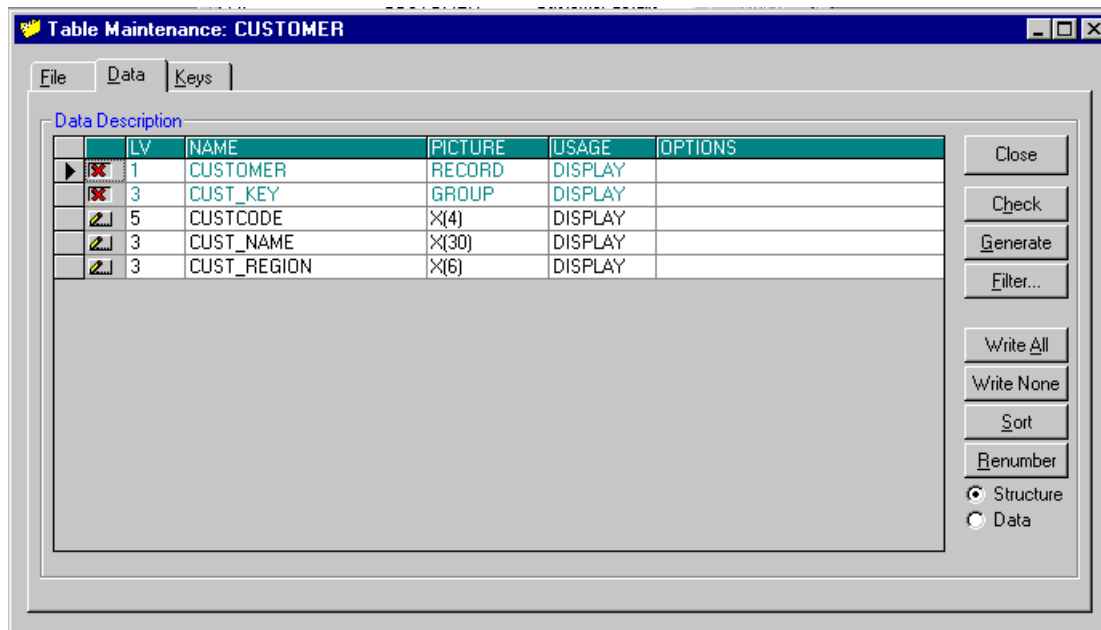
- **File**

This property page contains details of the name, organization type (for instance, indexed or indexed variable), and directory of the **physical** file. It also allows an expression to be entered that is used by the U/SQL Server to differentiate between multiple record types in the same physical file.

The following sections deal with these three property pages, Data, Keys and File, in this order.

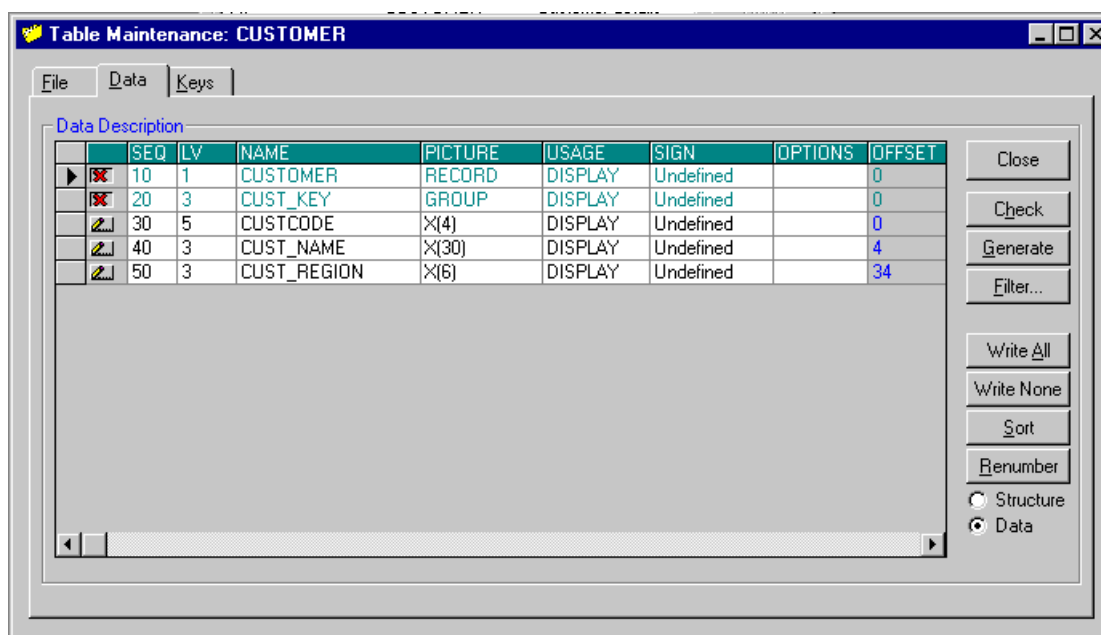
Step 9: Manipulate the Data

Select the **Data** tab. The Data Description property page is displayed:



By default the main **Structure view** of the data items is displayed.

Select the **Data** radio button to obtain the alternate COBOL view of the data items:



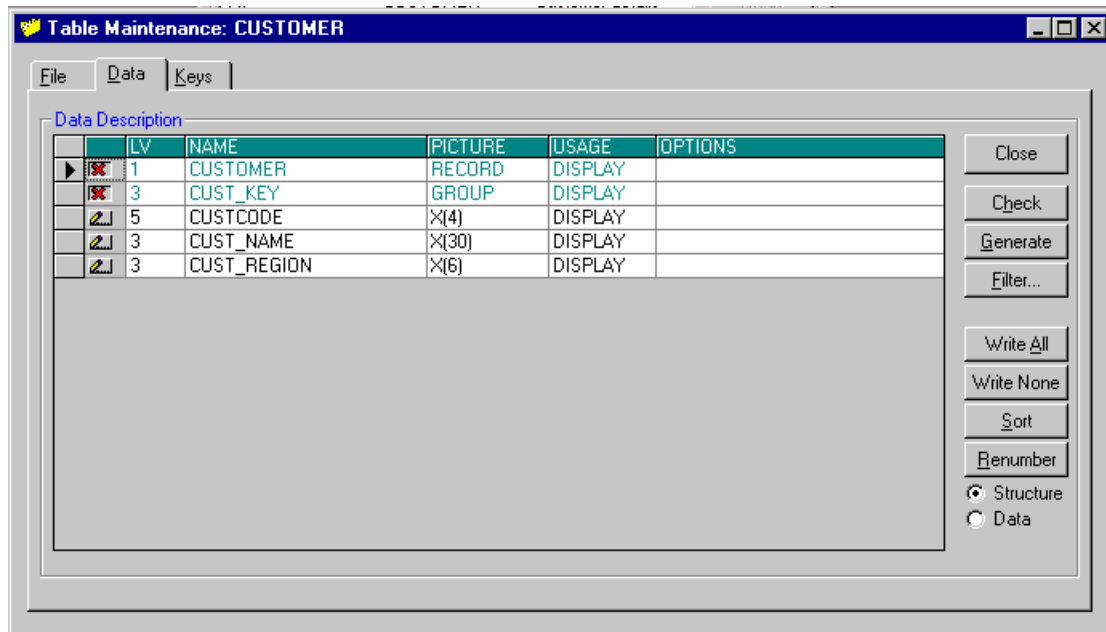
In most cases, you will work using the **Structure** view of the data items, as it is clearer and contains only the information you need to consider.

The above example shows details of the two views, the **Structure** and the **Data**, for the simple Data Description of the CUSTOMER table.

The forthcoming sections consider each of the items in the two views, in turn.

- [Structure view](#)
- [Data view](#).

Structure View



You will normally work with this **structure view** of the fields. The following sections explain each of the columns in the table:

- [Fields to be Written to the Dictionary](#)
- [LV](#)
- [NAME](#)
- [PICTURE](#)
- [USAGE](#)
- [OPTIONS.](#)

Fields to be Written to the Dictionary



The first column has either a pen icon indicating that the details will be written to the dictionary or an X indicating that they will not. You toggle between these states by clicking on the required element.

By toggling fields on or off, you decide which fields you want to include in the logical table that is visible to ODBC-enabled products. This allows you to create SQL views of the data that differ from what is actually present on the file.

Any field that forms part of an index must be written to the table.

Note: *COBOL 01 level Record and Group Items are always marked X and are not written to the UDD. Only elementary-items can be written to the dictionary, that is, items that have a picture.*

You can use the Write All or Write None buttons to set either all fields to be subsequently written to the UDD or none at all. These facilities can be a useful short-cut for establishing the fields that you want to see in the final table that can be accessed via the ODBC-enabled products.

COBOL Level (LV)

LV is the COBOL-level indicator of each field, in the COBOL FD, and has no relevance to U/SQL. It is included for reference only.

NAME

This is the name of the field (or **column**, as it is called in relational terminology) that will be used in all SQL statements. It is very important to make these names as meaningful as possible.

Note: *Column names that are the same in two tables are automatically joined by most ODBC-enabled products which makes these products easier to use. Having CUSTCODE in one table and CODE in another, both representing the customer code, would not automatically join together, while making them both CUSTCODE would.*

Each column name is normally limited to the ODBC compliant 18 characters, however you can optionally allow a maximum of 30 characters. If you use the optional 30 characters, ensure the ODBC-enabled applications you propose to use are capable of accessing column names of this size.

From the **Data Dictionary** menu, select **Options** and set **Allow Long Names** to use up to 30 character column names.

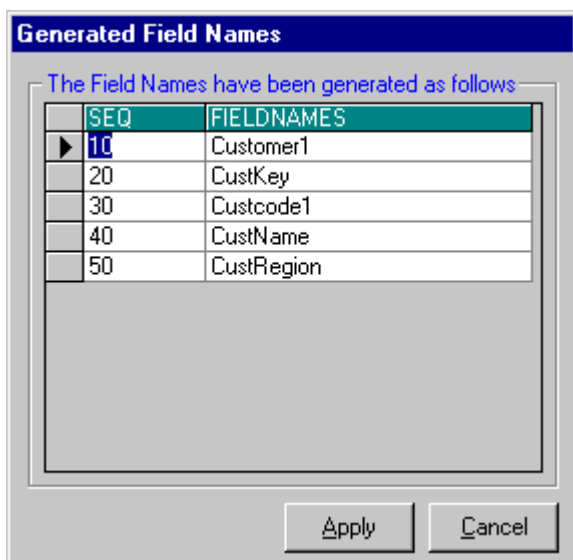
However, very often, COBOL field names are longer than 30 characters. You can check, by clicking the **Check** button that the names all satisfy the following conditions:

- are within either the 18 or 30 character limits
- are unique
- are not SQL keywords
- are not zero length
- are made up of 0-9, A-Z, a-z, _, with the first character alphabetic and with no blanks

You are notified of all field names that do not meet the above criteria and why. You can then modify the names.

In order to ensure the field names meet these criteria, you have two options:

- Click the **Generate** button. The **Generated Field Names** dialog box is displayed:



This dialog box automatically generates valid names based on the existing names. Click **Apply** to substitute these names for the originals. By double-clicking on any name, you can make further manual changes.

The corresponding names in the index Key fields will be updated as well.

Note: *This is the quickest way of creating valid dictionary entries.*

- Alternatively, click the **Filter** button. The **Field Name Filters** dialog box is displayed:



This facility allows you to:

- **Remove Text from Field Names** - Allows you to remove text from anywhere in the names.
- **Replace Text in Field Names** - Asks you to enter the Find What and the Replace With texts.
- **Add Text to Field Names** - Asks you what text you want added to either the beginning or the end of the names.

Choose the filter you want and then click **Next >>**.

Click the **Apply** button to make the changes.

Very often COBOL field names are of the form:

```
01 MAS-EMPREC.
   03 MAS-EMP-KEY.
       05 MAS-COMPANY          PIC 9(2) .
       05 MAS-NUMBER           PIC 9(5) .
       05 MAS-REC-TYPE         PIC X.
   03 MAS-ALT-KEY.
       05 MAS-LASTNAME         PIC X(20) .
```


where, in this case, there is a leading MAS-, which you may wish to remove automatically from all the fields as follows:

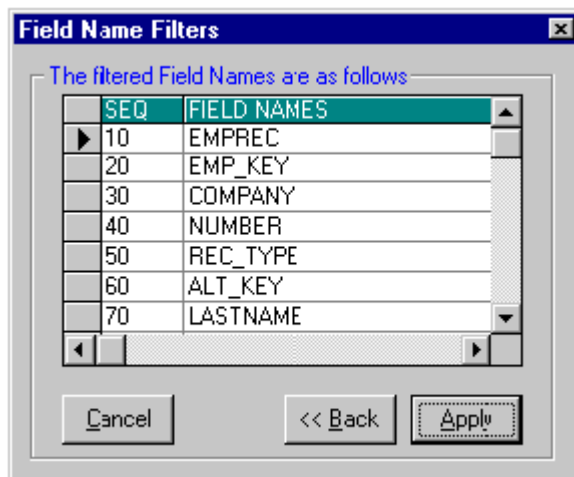
After the COBOL FD Converter has parsed this record, notice the '-' characters have already been converted to '_' characters:

| Data Description | | |
|------------------|----|--------------|
| | IV | NAME |
| ▶ | 1 | MAS_EMPREC |
| ▶ | 3 | MAS_EMP_KEY |
| ▶ | 5 | MAS_COMPANY |
| ▶ | 5 | MAS_NUMBER |
| ▶ | 5 | MAS_REC_TYPE |
| ▶ | 3 | MAS_ALT_KEY |
| ▶ | 5 | MAS_LASTNAME |
| ▶ | 3 | MAS_INITIALS |
| ▶ | 3 | MAS_SSNUMBER |
| ▶ | 3 | MAS_ADDRESS |
| ▶ | 5 | MAS_ADDRESS1 |
| ▶ | 5 | MAS_ADDRESS2 |
| ▶ | 5 | MAS_ADDRESS3 |
| ▶ | 5 | MAS_ZIP_CODE |

To remove the leading 'MAS_', click the **Filter** button. Select **Remove Text from Field Names** and then click **Next >>**. The following dialog box is displayed:



Enter the Text to be Removed and select **Start of Field Names**. Click **Next>>** and the filtered field names are displayed which you can apply to the table:



Note: *This process, of removing and adding text to field names, can be applied reiteratively to obtain the desired result.*

REDEFINES

Here is an example REDEFINES:

```
FD CODE-FILE.
01  CODE-REC
    03  CODE-FIELD                PIC X(16) .
    03  CODE-PARTS REDEFINES CODE-FIELD.
        05  NAME_PART             PIC X(10) .
        05  NUMBER                PIC 9(10) COMP-3.
```

This will be parsed and displayed as follows in the **Data Description**.

The UDD can hold the main field or its redefined component parts.

The FD parser assumes that all fields within a REDEFINES need to be added to the dictionary. For instance, in the example above, CODE_FIELD and its components CODE_NAME and NUMBER are all set with a Pen icon in the first column to indicate they will be written to the dictionary. Note that the CODE_FIELD and the CODE_NAME field will have the same byte offset in the physical file.

If you do not want any particular redefined field to be included in the dictionary, simply mark it by clicking on the first column to set it to X.

You may wish to have only records of a particular type defined by this table. These records would be selected depending on the value of one of the components of the REDEFINES. You can do this by including a record Expression in the File Description details, as described in section Step 11: File Details later in this chapter.

In the above example, you may wish to select only records where the NUMBER field contains values greater than zero. In this case, enter the record expression:

```
NUMBER > 0
```

in the **Expression** entry box on the **File** description details.

PICTURE

PICTURE is the COBOL picture of the field. The maximum size of the SQL alphanumeric data type is 254 bytes and is a general restriction of ODBC-enabled products. For fields bigger than this, the U/SQL Manager splits the field into a number of columns giving the new columns <NAME>1, <NAME>2 to <NAME>n, where <NAME> is the original name of the field.

When adding a column manually, it is best to enter the **PICTURE** first, followed by the other entries.

Note: *The combination of the entries in PICTURE and USAGE (see below) denotes each field's data type, which is automatically converted to the appropriate ODBC data type.*

USAGE

USAGE is the COBOL usage of the field. Click the USAGE field and a button appears. Click the button to display a menu of supported entries, for example DISPLAY, BINARY, COMP, and so on.

Normally, the entry is determined from the FD, but, if you are Adding a New Column (Field), you will need to specify its usage.

Note: The combination of the entries in **PICTURE** and **USAGE** denotes each field's data type, which is automatically converted to the appropriate ODBC data type.

OPTIONS

The **OPTIONS** column can contain a combination of "E", "N", "D" and "O", indicating that an **Expression**, **NULL Values**, **Date** or **Occurs** has been defined for any field in the record. In addition, **OPTIONS** can contain "V" indicating that the field is a **Virtual** column. If you want to declare any of these options for a particular field, click the **OPTIONS** entry for that field (which to start with will be blank) and a button appears. Click the button to display the following sub-menu:

Expression...

Virtual...

Null Values...

Date...

Occurs...

Then make your selection. The following sections describe how to set-up the following options:

- [Expression](#)
- [Null Values](#)
- [Date](#)
- [Occurs](#).

A **virtual** column can be defined in any table (record), where the value is determined from a **column expression**. A new column can be added to the table after the original record's FD details have been entered into the dictionary: refer to the section Adding a New Column (Field) in the next chapter, Modify a UDD.

Column Expression

As described in the [Expressions on Columns and Virtual Columns](#) section, an expression can be placed on any column (field) to change its value depending on the value of other columns in the same row (record).

If you want to place an expression on a column, click the **OPTIONS** entry for that column and a button appears. Click the button to display the submenu, and select **Expression....** The following dialog box is displayed:

Expression : CUSTOMER

Expression:
Enter a valid expression e.g:
IF crdb='DB' THEN -balance ELSE balance

IF FLAG='CR' THEN -SVALUE

Enter an underscore '_' to turn this feature off.

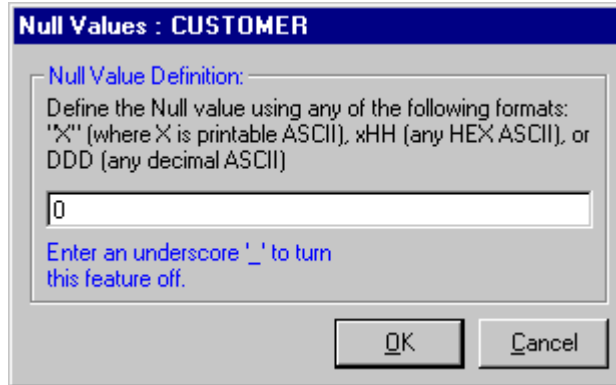
OK Cancel

Enter the expression and click **OK**.

Defining NULL Values

As described in the [Defining NULL Columns \(Fields\)](#) section, it is often useful if a particular value in a column (field) can be considered the SQL **NULL** value.

If you want to define NULL Values on a column, click the **OPTIONS** entry for that column and a button appears. Click the button to display the submenu, and select the **Null Values** command. The following dialog box is displayed:

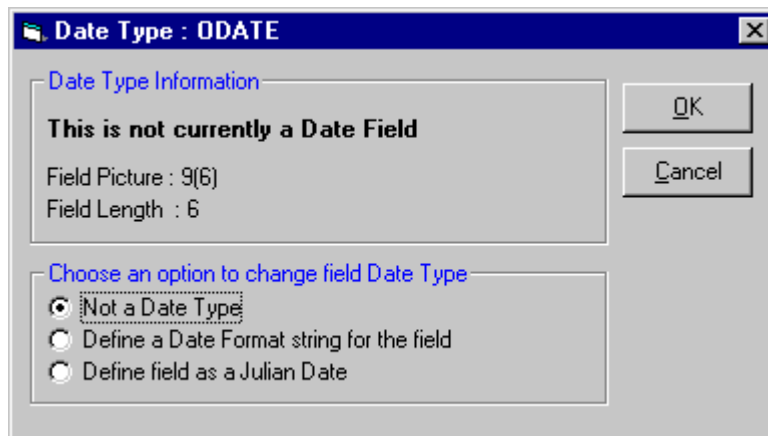


Enter the NULL value to be used and click **OK**.

SQL Date Formats

U/SQL is capable of manipulating a broad range of physical date formats, which it maps to the ODBC SQL Date data type (CCYYMMDD) that then allows date formatting and sorting by ODBC-enabled products.

If the field is known to be a date, click the **Options** entry and a button appears. Click the button to display the submenu, and select the **Date...** command. The **Date Type** dialog box is displayed:

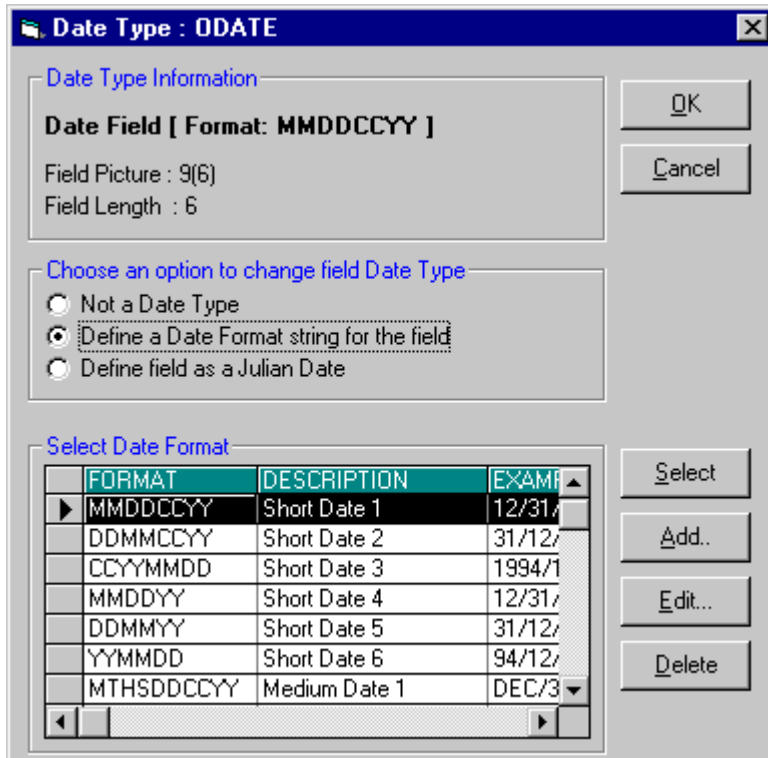


The field is initially defined as **This is not currently a Date Field**. The following additional options are available:

- [Define a Date Format string](#)
- [Define field as a Julian Date.](#)

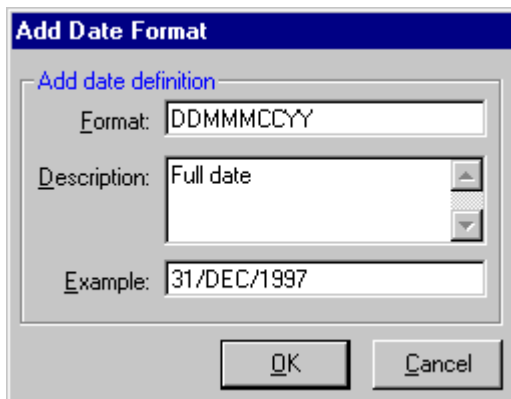
Date Format String

If you click **Define a Date Format string** for the field, the dialog box is expanded and appears as below:



This dialog box allows you to:

- Pick one of the existing formats, by highlighting it and clicking the Select button or by double clicking on it, or
- Enter your own format. Click the **Add...** button. The **Add Date Format** is displayed:



The **Date Format String** must be less than or equal to the length of the field it is defining. You are warned if this not the case. Characters that are allowed in the Date Format String are:

- For day: D
- For month: M
- For year: Y
- For century: C
- For special formats: I, F and R - see below

All other characters are assumed to be separators, such as '/', or '-'.

FixedDateOffset

The optional directive, **FixedDateOffset=nn**, allows you to define a 'cut-off' year below which dates with two digit years are considered to be 20nn. The value of 'nn' may be 0 to 99. Assume, nn=30, then any year 0 to 29 is considered 2000 to 2029 and any year 30 to 99 is considered to be 1930 to 1999.

Before using this directive you should check that existing date data does not come into the range prior to the offset date. Obviously, setting the directive will 'move' any such dates forward by 100 years. To check whether any records exist with dates in this range issue the following query:

```
select count(*) from <table> where <date> <"19nn-1-1"
```

If the count is zero then no such records exist.

For further details on FixedDateOffset and other directives, refer to the [ODBC.INI Directives](#) section.

Note: *Illegal dates are treated as NULL and a warning is written to the log file.*

Special Date Format Strings

The following Date Format strings can be added if required:

- **Format string - IYYMMDD**

A common storage requirement for dates is to store in the form YYMMDD. Consider the following COBOL type structure:

```
01 W-DATE.
   03 W-YY      PIC 99.
   03 W-MM      PIC 99.
   03 W-DD      PIC 99.
```

This is populated by a statement such as:

```
ACCEPT W-DATE FROM DATE.
```

With some compilers, the fact that W-DATE is a 'group field' and hence by implication alpha-numeric will prevent this syntax from being accepted, however if W-DATE is defined as PIC 9(6) then it is an acceptable picture clause. An equally acceptable alternative is to define the field as PIC 9(6) COMP.

When considering year 2000 compliance, and indeed any date after 31st December 1999, the six digit format will (potentially) store the dates out of sequence. For example 31-Dec-1999 is 991231 and 01-Jan-2000 is 000101.

As 101 < 991231, sorted dates will be out of sequence.

One way of extending the life of the mechanism is to allow the 'year' component to exceed 99, that is year 100 = 2000, 101 = 2001 and so on. This can be done by extending the digits in the picture to 9(7), however this will generally change the underlying data storage requirement, and this will probably require the data to be re-formatted.

There are exceptions to this rule which can be exploited. For example a PIC 9(6) COMP requires 3 bytes or 24 bits of storage, and the largest number that can be stored is 8388607 (although some systems may set other internal limits).

Care needs to be taken, however, when exploiting this 'overflow' situation. With Micro Focus COBOL, the "NOTRUNC" directive must be enforced on all programs. The equivalent ACUCOBOL compiler switch is -Dz.

Failure to enforce these directives on all programs will result in a 'move' truncating the date and losing the overflow digits which are being used to hold the century marker.

If this method is used to store dates after 1999, U/SQL Adapters supports an extended 'picture' to decode the year. The date format string YYMMDD will decode the traditional 6 digit year in the range 1900 to 1999.

The date format string **IYYMMDD** will decode the overflow digit so that dates from 1900 to 2099 can be processed, either from a PIC 9(7) data item, or from a PIC 9(6) COMP data item where the data is able to overflow the picture within the bounds of the underlying data storage.

- **Format string - IYYDDD**

- 6 byte, un-terminated alphanumeric date.
- I represents century , and it is '0' for 1900 and '1' for 2000.
- YY represents a standard two-digit numeric character representation of the year.
- DDD represent a julian date.

- **Format string - C1YYDDD**

- 6 byte, un-terminated alphanumeric date.
- C1 represents century , and it is '1' for 1900 and '2' for 2000.
- YY represents a standard two-digit numeric character representation of the year.
- DDD represent a julian date.

- **Format string - RCRYMRD or RYMRD**

The RC, RY, RM & RD tokens are used to make up nines compliment dates - either RCRYMRD or RYMRD.

A date with date format string of **RCRYMRD** has a stored value of (99999999 - CCYYMMDD) and one stored as **RYMRD** has a value of (999999 - YYMMDD).

For example, 1997-12-08 using **RCRYMRD** stores (99999999 - 19971208) = 80028791 and 1997-12-08 using **RYMRD** stores (999999 - 971208) = 028791.

- **Format string - FYMMDD**

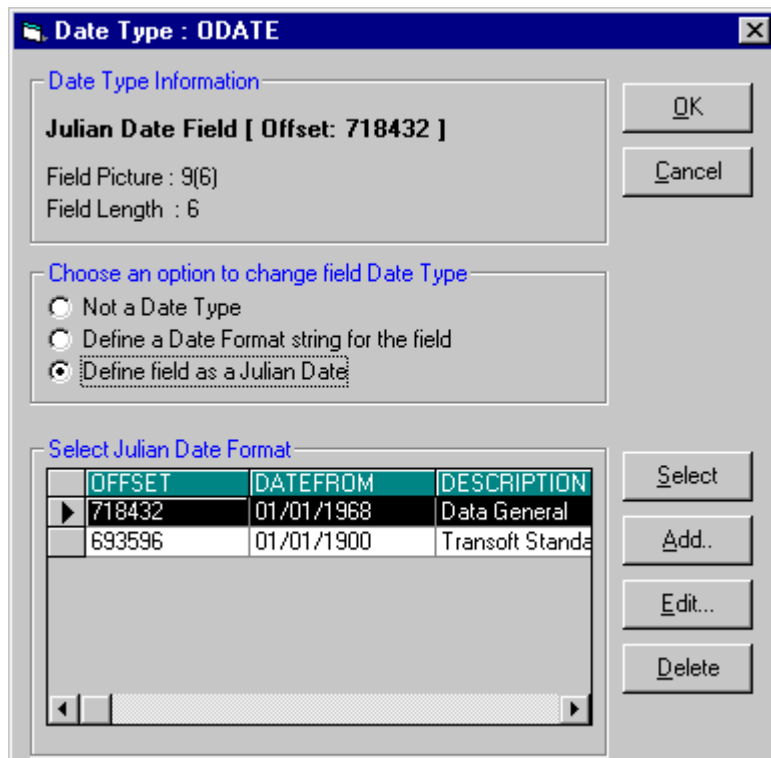
- 6 byte, un-terminated alphanumeric date.
- **F** is in the range of <space> (ASCII 32) to 'I' (ASCII 73) representing the decades 1740 to 2150.
- **Y** is in the range of '0' (ASCII 48) to '9' (ASCII 57) representing the least significant digit of the year.
- **MM** and **DD** represent a standard two-digit numeric character representation of the month and day.

- **Format string - FCRMRD**

- 6 byte, un-terminated alphanumeric date.
- **F** is in the range of <space> (ASCII 32) to 'I' (ASCII 73) representing the decades 2150 down to 1740. The character value is calculated by subtracting the Frontier format decade value from 105.
- **C** is in the range of '0' (ASCII 48) to '9' (ASCII 57) representing the nines compliment of the least significant digit of the year. The character value is calculated by subtracting the digit's ASCII value from 105.
- **RM** and **RD** represent a two-digit numeric character representation of the month and day in nines compliment format.

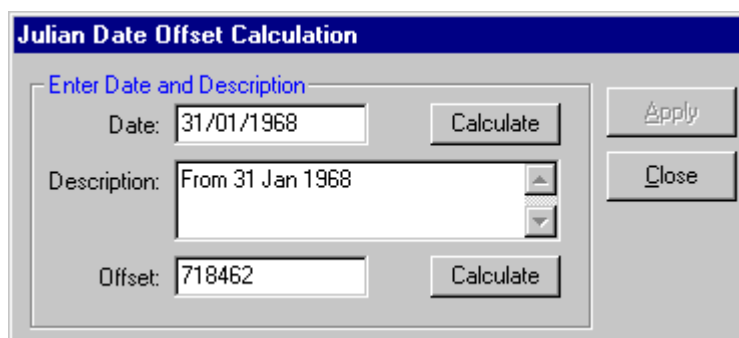
Julian Dates

Click the **Define field as a Julian Date** radio button on the **Date Type** dialog box, and the dialog box is expanded as is shown below:



This dialog box you to:

- Pick one of the existing formats, by highlighting it and clicking **Select**, or by double clicking it
- Enter your own format. Click the **Add...** button. The **Julian Date Offset Calculation** dialog box is displayed:



You either enter the commencement **Date** for your Julian date and click **Calculate**, which calculates the Julian **Offset**, where 1 is 1st January 0001 (the base or epoch date). Alternatively, enter the **Offset** number of days (if you know it) and click **Calculate**, which calculates the commencement **Date**.

OCCURS

Note: A detailed discussion on OCCURS is provided in the [Handling Data Arrays](#) section.

U/SQL can handle differing COBOL OCCURS.

[OCCURS can be flattened out into individual unique fields](#), or maintained as any combination of the following:

- [Repeating groups](#)
- [Parallel OCCURS](#)
- [Nested OCCURS](#)
- [OCCURS DEPENDING ON](#) (and, optionally, for Micro Focus EXTfH, ODOSLIDE).

Each of these is described below.

Note: Clicking the **Write All** button does not write OCCURS to the UDD, by default.

OCCURS Flattened out into Individual Unique Fields

Consider the following example of the ADDRESS that OCCURS three times:

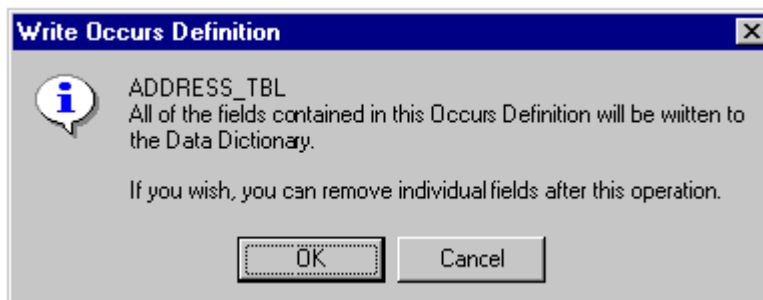
```
03 ADDRESS      PIC X(30)
                OCCURS 3 TIMES.
```


For this type of OCCURS, you would want to flatten it out into individual fields, one for each address line, to be the equivalent of:

```
03 ADDRESS1     PIC X(30)
03 ADDRESS2     PIC X(30)
03 ADDRESS3     PIC X(30)
```

When the COBOL FD Converter parses the FD, it automatically assumes that the OCCURS is to be a repeated group. It therefore inserts an **OCCURS table** field and a **subscript** field, which have the name of the first field in the OCCURS with **_TBL** and **_IDX** suffixes. In this example, these are ADDRESS_TBL and ADDRESS_IDX. In the OPTIONS column for ADDRESS_TBL, there is an entry "O", indicating an OCCURS.

Notice that none of the fields will currently be written to the dictionary, as they are all set to **X** in the first column. To include the OCCURS in the details to be written to the dictionary, click on the first column position (that is, the X) of the **OCCURS table** field, for example, ADDRESS_TBL. The following message is displayed:



Clicking **OK** will cause all the fields of the OCCURS to be available to be written to the dictionary, denoted by the  icon, but a repeated group is still assumed.

Click the **OPTIONS** column for the ADDRESS_TBL field and a button will appear. Click the button, and select **Occurs...** to display the **Occurs** dialog box:

Select the **Individual Fields** button, then click **OK**. The OCCURS is then re-written as individual unique fields, as shown below:

At this point you have finished working with the Data or fields definition, unless you need to deal with **Repeating Groups**, **Parallel OCCURS**, **Nested OCCURS**,

or **OCCURS DEPENDING ON** functions. These topics are covered in the following sections.

Repeating Groups

Consider the following example FD with the JOBS-GROUP occurring 12 times:

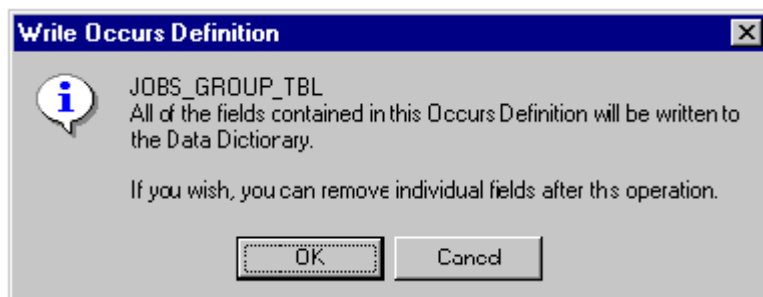
```

01 JOBS-RECORD.
   03 JOBS-KEY.
       05 JOBS-NO                PIC X(6) .
   03 JOBS-NAME                  PIC X(30) .
   03 JOBS-DESCRIPTION           PIC X(50) .
   03 JOBS-GROUP OCCURS 12 TIMES.
       05 JOBS-ACCD              PIC S9(10)V9(2)
                                   SIGN TRAILING SEPARATE.
       05 JOBS-PAID              PIC S9(10)V9(2)
                                   SIGN TRAILING SEPARATE.

```

The JOBS-GROUP OCCURS can be considered as 12 rows in a relational table, that is, for each JOBS-RECORD there will be 12 repeating rows as a group. When the COBOL FD Converter parses the FD, it automatically assumes that the OCCURS is to be a repeated group and inserts an **OCCURS table** field and a **subscript** field, which have the name of the first field in the OCCURS, with **_TBL** and **_IDX** suffixes. In this example, these are JOBS_GROUP_TBL and JOBS_GROUP_IDX respectively.

Notice that none of the fields will currently be written to the dictionary, as they are all set to **X** in the first column. To include the OCCURS in the details to be written to the dictionary, click on the first column position (that is, the X) of the **OCCURS table** field, for example, JOBS_GROUP_TBL. Then click again and the following message is displayed:



Click **OK** and all the fields of the OCCURS will be available to be written to the dictionary.

For a repeated OCCURS group, the **subscript** field is effectively an extra index key which, in this case, will have a value of 1 through 12 for each occurrence row for each JOBS_RECORD.

When written to the dictionary, this extra **subscript** field is visible to ODBC-enabled products, so it is advisable to change it to something more meaningful to you, such as MONTH_NO in this example. To do this, click the **OPTIONS** column for the JOBS_GROUP_TBL field and a button will appear. Click the button and select **Occurs....** The **Occurs** dialog box is displayed:

The image shows a dialog box titled "Occurs : JOBS_GROUP_TBL". It has two main sections. The top section, "OCCURS Defined As", contains two radio buttons: "Repeating Group" (which is selected) and "Individual Fields". Below this is a text box labeled "Index Field Name:" containing the text "MONTH_NO". The bottom section, "Fixed Length", contains two labels: "Maximum Size: 12" and "Minimum Size: 1". On the right side of the dialog, there are three buttons: "OK", "Cancel", and "Table".

This index field is also added as the last component of the index key. Ensure that this change of name is also reflected in the index: refer to the [Step 10: Keys Definition](#) section.

The OCCURS, JOBS_GROUP_TBL is defined as a **Repeating Group**. You can change the **Index Field Name**, that is, the JOBS_GROUP_IDX subscript field, to, say, MONTH_NO to be more meaningful. Click **OK**.

Note:

When you perform the following SQL query:

```
Select * from JOBS_RECORD;
```

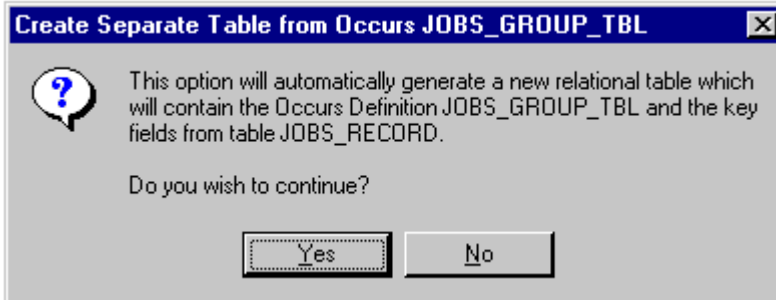
on this JOB_RECORD table via your ODBC-enabled product, for each JOBS_RECORD on file you will have 12 rows displayed. The same values of JOBS_NO, JOBS_NAME and JOBS_DESCRIPTION will be repeated in each of the 12 rows. The new MONTH_NO column will have values of 1 through 12.

Define Another Logical Table

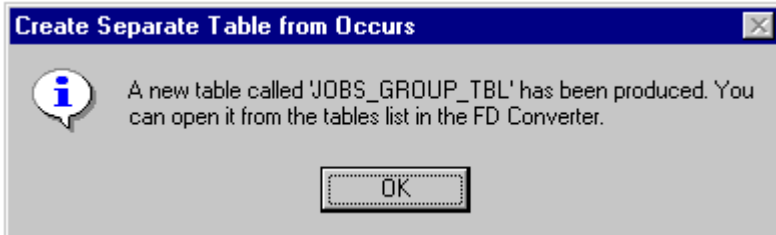
An alternative to maintaining the OCCURS as a repeating group within the existing record is to define another logical table that just contains the key information of the main record together with the repeating group.

This is an identical copy of the dialog box shown in the first image. It is titled "Occurs : JOBS_GROUP_TBL" and contains the same options: "Repeating Group" (selected), "Individual Fields", "Index Field Name: MONTH_NO", "Fixed Length" section with "Maximum Size: 12" and "Minimum Size: 1", and buttons for "OK", "Cancel", and "Table".

Using our JOBS_RECORD, from the Occurs dialog box, click the **Table** button. The following message is displayed:



Click **Yes** to create a new logical table called JOBS_GROUP_TBL and put it into the **COBOL FD Converter** dialog box:



The U/SQL Manager returns you to the **Occurs** dialog box. After clicking **OK** on the **Occurs** dialog box, you will still be in the original JOBS_RECORD details. Ensure that the OCCURS fields are 'switched off', that is, marked by an **X** in the first column, so that they will not be written to the dictionary. If they are not marked with an X, click on the pen icon against the JOBS_GROUP_TBL entry. Thus JOBS_RECORD only contains the view of the fields JOBS_NO, JOBS_NAME and JOBS_DESCRIPTION.

If you exit from the JOBS_RECORD details using the **Close** button, the **COBOL FD Converter** dialog box is displayed:

Open the new table JOBS_GROUP_TBL from the **COBOL FD Converter** dialog box, and you will see again all the fields from the original JOBS_RECORD table. However, now the non-key fields are set to be excluded from updating the dictionary, as shown below.

Check that all the key fields are to be written to the dictionary.

At this point you have finished working with the **Data** or fields definition of the two tables JOBS_RECORD and JOBS_GROUP_TBL. You will probably want to rename this latter table, as previously described.

Unless you need to deal with [Parallel OCCURS](#), [Nested OCCURS](#) or [OCCURS DEPENDING ON](#) functions, you can skip to the [Reference View of the Data Fields](#) section or the [Step 10: Keys Definition](#) section.

Parallel OCCURS

It is possible to have more than one OCCURS defined as repeating groups within the same logical table, known as parallel OCCURS, but they must all have the same number of occurrences. The COBOL FD Converter automatically inserts a single subscript field, called INDEX1, before the first of the repeating groups. You can change this name to one more meaningful to you. Each repeating group will also have an OCCURS table field that has the name of the first field in the OCCURS, with _TBL extension.

For example, the JOB_RECORD below has three repeating groups each of 12 occurrences. The subscript field would appear before JOBS_ACM and would be automatically created as INDEX1, with the three OCCURS table fields. In this example, these are JOB_ACM_TBL, JOB_PYM_TBL and JOB_BLM_TBL, respectively.

This subscript field (INDEX1) is effectively an extra index key which, in this case, will have a value of 1 through 12 for each occurrence row for each JOB_RECORD.

```

01  JOB-RECORD.
    03  JOB-KEY.
        05  JOBS-NO                PIC X(6) .
    03  JOB-GROUP.
        05  JOB-ACCD                PIC S9(10)V9(2)
                                SIGN TRAILING SEPARATE.
        05  JOB-ACM                 PIC S9(10)V9(2)
                                SIGN TRAILING SEPARATE
                                OCCURS 12 TIMES.
        05  JOB-PAID                PIC S9(10)V9(2)
                                SIGN TRAILING SEPARATE.
        05  JOB-PYM                 PIC S9(10)V9(2)
                                SIGN TRAILING SEPARATE
                                OCCURS 12 TIMES.
        05  JOB-BILL                PIC S9(10)V9(2)
                                SIGN TRAILING SEPARATE.
        05  JOB-BLM                 PIC S9(10)V9(2)
                                SIGN TRAILING SEPARATE
                                OCCURS 12 TIMES.

```

The COBOL FD Converter produces the following Data Description display for this JOB_RECORD.

You must decide how you wish to treat each of the OCCURS. In this case, you will probably keep them in the one table as parallel OCCURS. Alternatively, you can 'mix & match' the differing types of OCCURS. In this example:

Each OCCURS could be a separate logical table, or

One or more of them could be flattened into individual unique fields within the same overall table, or

There could be a combination of the two.

At this point you have finished working with the Data or fields definition of the above tables, unless you need to deal with [Nested OCCURS or OCCURS DEPENDING ON](#) functions. These topics are covered in the following pages, or you can skip to the [Reference View of the Data Fields](#) section or the [Step 10: Keys Definition](#) section.

Nested OCCURS

COBOL allows multiple nesting of OCCURS. U/SQL Adapters supports this with up to 10 levels of nesting. When the COBOL FD Converter parses the FD, it inserts an **OCCURS table** field and a **subscript** field, at each level of nesting, which have the name of the first field in the OCCURS with **_TBL** and **_IDX** extensions.

The **subscript** field is effectively an extra index key at each level of nesting. You can change the generated names to be more meaningful to you.

In the example below, two **subscript** fields, JBS_ACCD_IDX and JBS_ACM_IDX, are created.

```

01  JBS-RECORD.
    03  JBS-KEY.

```

```

05 JBS-NO          PIC X(6) .
03 JBS-ACCD       OCCURS 12 TIMES.
05 JBS-ACM       PIC S9(10)V9(2)
SIGN TRAILING SEPARATE
OCCURS 6 TIMES.

```

Note: Each row at each level of nested selection will include all the preceding levels' fields repeated. In the case above, for each unique JBS_KEY, 72 rows will be retrieved at the second level of nesting.

You must decide how you wish to treat each of the OCCURS. You can keep it as one table as **nested OCCURS**. Alternatively, you can 'mix & match' the differing types of OCCURS. In this example:

- Each OCCURS could be a separate logical table (although you cannot have an 'inner' OCCURS in a separate table without including its parent OCCURS), or
- One or both of them could be flattened into individual unique fields within the same overall table, or
- There could be a combination of the two.

At this point you have finished working with the Data or fields definition of the above table(s). Unless you need to deal with the [OCCURS DEPENDING ON](#) function, you can skip to the [Reference View of the Data Fields](#) section or the [Step 10: Keys Definition](#) section.

OCCURS DEPENDING ON

COBOL supports the concept of an OCCURS that, for each record, has a variable number of occurrences. This is determined by the OCCURS DEPENDING ON clause. Take the following example:

```

01 CUSTREC.
03 CUST-KEY.
05 CUST-ID          PIC 9(6) .
03 CUST-NAME       PIC X(30) .
03 SALES-COUNT     PIC 9.
03 SALES OCCURS 1 TO 12 DEPENDING ON SALES-COUNT.
05 SALES-ACTUAL   PIC S9(10)V9(2) .
05 SALES-BUDGET   PIC S9(10)V9(2) .

```

This FD shows that the SALES occurs between 1 and 12 times and is DEPENDING ON the value of SALES-COUNT.

When the COBOL FD Converter parses the FD, it inserts an **OCCURS table** field, which have the name of the first field in the OCCURS with **_TBL** and **_IDX** extensions. You can change this subscript name to one more meaningful to you.

Notice that the PICTURE column in the **Data Description** field has an **OCCURS 1 to 12** against it, in this example, the SALES_TBL field.

To include the OCCURS in the details to be written to the dictionary, click the first column position (that is, the X) of the **OCCURS table** field, for example SALES_TBL.

Then click the **OPTIONS** column for the SALES_TBL field and a button will appear. Click the button and select **Occurs...**. The **Occurs** dialog box is displayed:

Occurs : SALES_TBL

OCCURS Defined As

Repeating Group

Individual Fields

Index Field Name:

SALES_IDX

Variable Length

Maximum Size: 12

Minimum Size: 1

Depending On:

SALES_COUNT

OK

Cancel

Table

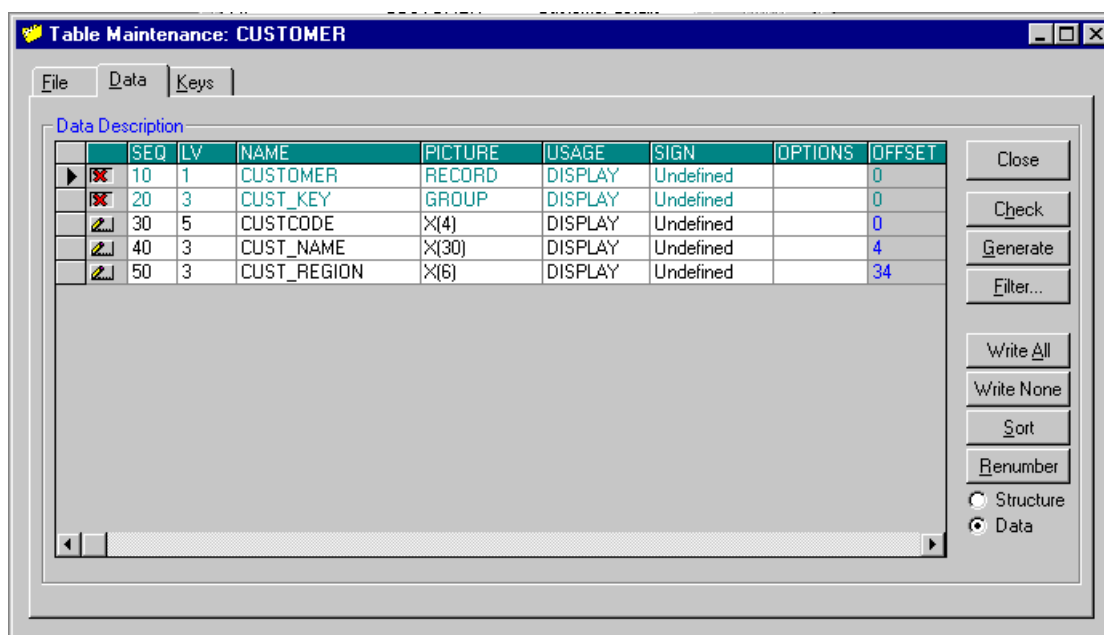
You will see that the OCCURS, SALES_TBL is defined as a **Repeating Group** which is **Variable Length**, with a Maximum Size of 12 and a Minimum Size of 1, and where the **Depending On** field is SALES_COUNT. You can change the Index Field Name, that is, the **subscript** field SALES_IDX, to, say, MONTH_NUMBER to be more meaningful to you.

For Micro Focus COBOL, the compiler directive **ODOSLIDE** is supported. Refer to the [Micro Focus COBOL Compiler Directives](#) section.

At this point you have finished working with the Data or fields definition of the table. Proceed to the [Reference View of the Data Fields](#) section, or the [Step 10: Keys Definition](#) section.

Data View

Reference View of the Data Fields



Select the **Data** radio button to view the COBOL reference view of the Data Description of the field items making up the record.

The additional columns of information not already described in the [Structure View](#) section, are detailed below.

SEQ

You can change the sequence number of the fields and then use the **Sort** button to re-sequence the field names, for example, to give better grouping of the field items as they will appear to the ODBC-enabled product user. The byte offsets do not change.

You can also re-number the sequence numbers, in steps of 10, using the **Renumber** button.

The following columns are obtained from the COBOL FD and must not be changed:

SIGN

Click the SIGN field and a button appears. Click the button to display a menu of supported entries, for example, LEAD, LEAD SEP, TRAIL, and so on.

Normally, the SIGN entry is determined from the FD, but, if you are Adding a New Column (Field), described in the next chapter, Modify a UDD, you will need to specify the SIGN entry.

OFFSET

Byte offset of the field within the record.

LEN

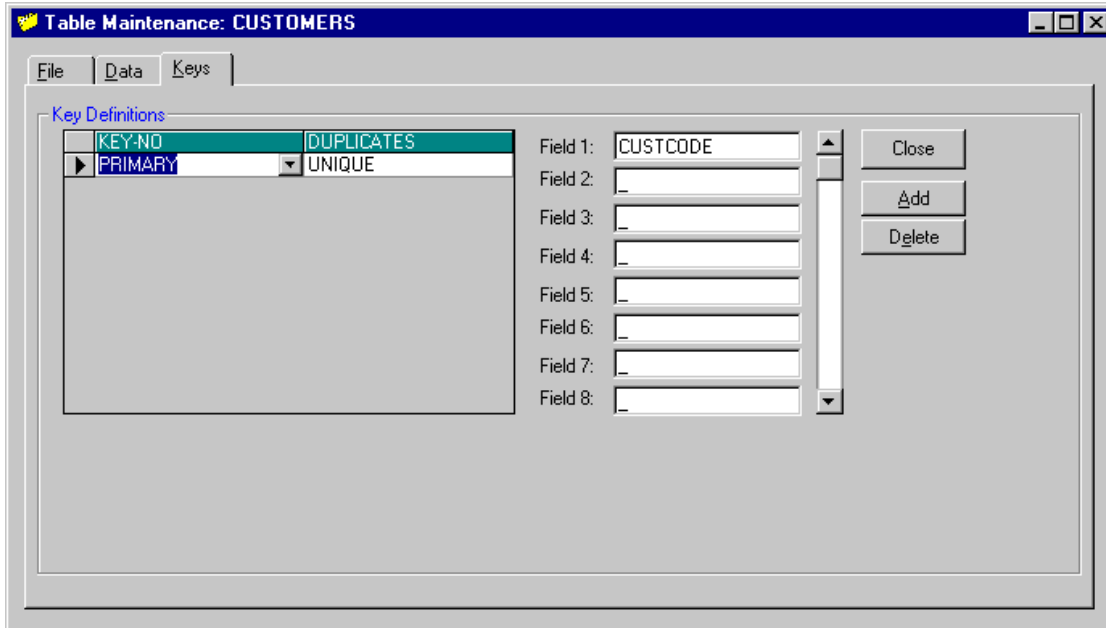
Length of field in bytes. The maximum size of the SQL alphanumeric data type is 254 bytes and is a general restriction of ODBC-enabled products. For fields bigger than this, the COBOL FD Converter will automatically split the field into a number of columns giving the new columns <NAME>1, <NAME>2 to <NAME>n, where <NAME> is the original name of the field.

DEC

Number of decimal places.

Step 10: Keys Definition

After you have defined the details of the Data or fields definition, select the **Keys** tab. The **Key Definitions** property page is displayed:



If COBOL **SELECT** statements are supplied, the COBOL FD Converter creates all the **Key Definitions** automatically and you will not usually make any changes. If subscript fields have been generated for OCCURS then these are automatically added to the end of the key components. If the name of a subscript field is changed then ensure that the index entry is also changed.

If your COBOL FDs qualify data names in the specification of any index, that is:

```
RECORD KEY IS <data-name> OF <record-name>
```

then the U/SQL Manager may not correctly identify this key field.

It is therefore important to check that the entries for each index contain all the key components. Failure to do so will result in performance degradation as the index will not be used in certain cases, when it should be.

You can check the contents of your indexes using the appropriate COBOL utility:

- ACUCOBOL - utility **vutil**
- Micro Focus COBOL - utility **fhinfo**

Refer to your COBOL documentation for further details.

Manually Adding an Index

If the **SELECT** statements are not supplied, the Key Definitions must be manually entered.

Click the **Add** button, which creates an entry under the **KEYNO** and **DUPPLICATES** headings, as **ALT-x** and **UNIQUE**, where 'x' is the next index number. If you want to change the **KEYNO**, click the field and a button appears. Click the button to display a list box of **PRIMARY, ALT-1** to **ALT-64**, allowing up to 65 indexes for each table (record). Make your selection.

If the index you are defining allows duplicates, click the **UNIQUE** field to display a button. Click this button, then select **DUPPLICATES**.

At this point, you add the fields 1 to 64 that make up this index key. Remember to remove the '_' character from each field used. Should you try to add a field (column) name that does not exist in the table, then the **Invalid Field Name** message is displayed.

ACUCOBOL - Order of alternate indexes

The U/SQL Manager orders secondary indices in the same way that ACUCOBOL does. This means that index information is gathered from the SELECT information in the definition file, it is then sequenced using the FD section to ensure that the ordering of the indices matches the physical ordering of the fields in the data file.

This checking is limited to the first part of any secondary index, therefore in the situation where two indices have matching first parts, their order will be determined by the **SELECT** statement and not by the FD information.

Deleting an Index

To delete an index entry, click on the appropriate index and click the **Delete** button.

At this point you have finished working with the **Keys definition** of the table. Proceed to the [Step 11: File Details](#) section .

Step 11: File Details

After you have defined the details of the **Data** or fields definition and the index **Keys** definition for a particular table, select the **File** tab. The **File Details** property page is displayed:

The screenshot shows a window titled "Table Maintenance: CUSTOMER" with three tabs: "File", "Data", and "Keys". The "File" tab is active. It contains several sections:

- File Details:** Description: "Customer details", Name: "customer", Directory: (empty), Organisation: "INDEXED" (dropdown).
- Record Details:** Length: "40", Min Length: "40", Max Length: "40".
- Data Filter:** Expression: (empty).
- Parsing Options:** Storage Mode: "COMP6(2)".
- Character Set:** Radio buttons for "ASCII" (selected), "EBCDIC", and "Sign EBCDIC".

A "Close" button is located in the top right corner of the dialog.

This property page is used to enter details of the name of the physical file on the disk that relates to this logical table. Remember, if you have multiple 01 records in a COBOL FD, you will have separate logical tables for each record type but they will have the same physical file details. They are differentiated by having an expression that allows each record type to be separately selected.

In the example below, the FD for the physical file EMPFILE has two 01 records, Employee Master and Employee Salary records. There will be two logical tables defined, both with the same file details, apart from the **Expression**. In the case of the Employee Master table, these records are determined by the expression `MAS_REC_TYPE="M"`.

The **File Details** are entered as follows:

- **Description:** type in the description you want for the file. This is only used for information and is not available via any ODBC-enabled product. The minimum entry must be an underscore, '_'.
- **Name:** this is the physical name of the file and is picked up automatically by the COBOL FD Converter, if available in the FD. For Single-tier U/SQL you can click the **Name...** button to browse for the file and its directory.
- **Directory:** this is the path of the physical file.

It is recommended that this is left 'blank'. In this case the filename will be searched for down the **Searchlist=** directive. For Multiple-tier, this is set as a U/SQL Server directive; For Single-tier, this set as an INI directive. See the [INI Directives](#) section for further details.

Note: If you have already placed directory paths into various tables and want to now remove them, then instead of using the U/SQL Manager to remove each directory individually you can use the following SQL statement in the Interactive U/SQL utilities, [Win U/SQLi](#) (on the client PC) or [usqli](#) (on the UNIX server) to globally remove all directory paths:

```
update cobol_table set pathid=0;
```

However, the Directory entry can be one of the following:

- The full path name to the directory where the file is located, for example, **/usr/datafiles** (UNIX), **C:\DATAFILES** (Windows).
- The directory relative to where the U/SQL Server software resides. For example, assume the UNIX based U/SQL Server is started in **/usr/usqls/bin** and the data file is in **/usr/datafiles**, the relative path is **../../datafiles**.

Note: *If the directory entries are of this type, there must be an entry of 'Directory=..' in the UNIX U/SQL Server configuration file, [usqlsd.ini](#).*

- The directory relative to the base directory where the data files are located. For example under UNIX, if the base directory where the data files are stored is **/usr/datafiles** and a particular data file is in the directory **/usr/datafiles/sales**, then its relative path is **sales**.

Note: *If the directory entries are of this type, there must be an entry of 'Directory=/usr/datafiles' in the UNIX U/SQL Server configuration file, [usqlsd.ini](#).*

- **Organization:** this is the organization of the file - Indexed, Relative, Sequential, Line Sequential, Variable Sequential, Variable Relative or Variable Indexed. Click the Combo box button to show the options.

The **Record Details** are entered as follows:

- **Length:** this is the byte length of the record. It is determined by the COBOL FD Converter ascertaining the field with the highest byte offset and adding the size of the field.

Ensure the record length is the same as that in the physical file and that your FD has not been truncated for any reason, for example, trailing FILLER has not been specified. Use your COBOL utilities to check the actual record lengths. If an FD has been truncated, the record length can be manually entered.

- **Min & Max Length:** if the record is variable length, then you are able to adjust both the **Min Length** and **Max Length** record lengths.

The **Data Filter** is entered as follows:

- **Expression:** this is any record expression, which can be a maximum of 254 characters, that is used to distinguish between multiple record types in the same physical file. The minimum entry must be an underscore, '_'. An expression will normally be associated with multiple 01 records in the same FD file: see the [Multiple 01 Records](#) section.

The **Parsing Options** are displayed:

- For Micro Focus COBOL, you may have previously set the compiler directive ODOSLIDE, indicating OCCURS Depending On Slide. ODOSLIDE affects data items that appear after a variable-length table in the same record; that is, after an item with an OCCURS DEPENDING ON clause but not subordinate to it. With ODOSLIDE, these items always immediately follow the table, whatever its current size; this means their addresses change as the table's size changes.

For Micro Focus COBOL, you are able to define various **Character Set** settings:

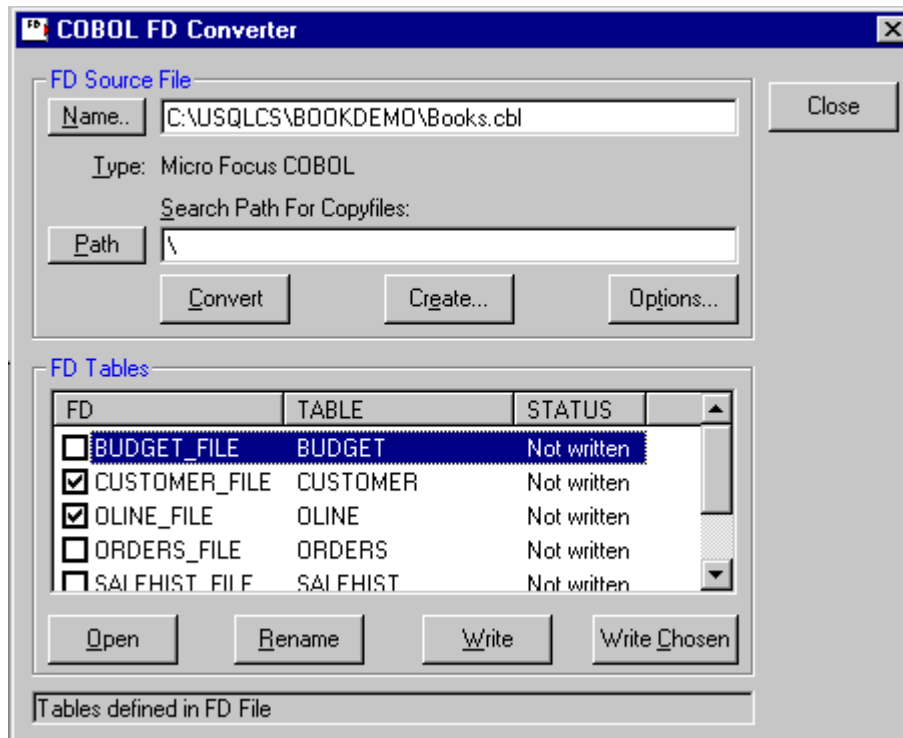
- **ASCII, EBCDIC and Sign EBCDIC.**

If there are multiple 01 records defined in the same physical file, each record type is described as a separate logical table by the COBOL FD Converter. You need to know how each record type can be uniquely distinguished from the others. This is usually determined by one or more fields having particular values. You must decide on the different names you wish to call each logical table and establish the differentiating values or expressions. For further information, see the sections [Dealing with Multiple 01 Records](#) and [Expression Handling Syntax](#).

Step 12: Write the Table Information to the Data Dictionary

After you are satisfied that you have set up the **Data** or field definitions, index **Keys** definitions and **File** definitions for the table you have been working on, as described in [Step 9](#), [Step 10](#) and [Step 11](#) respectively, click the **Close** button.

Next, click the name of the table, for instance CUSTOMER, in the FD Tables list in the **COBOL FD Converter** dialog box and click the **Write** button:



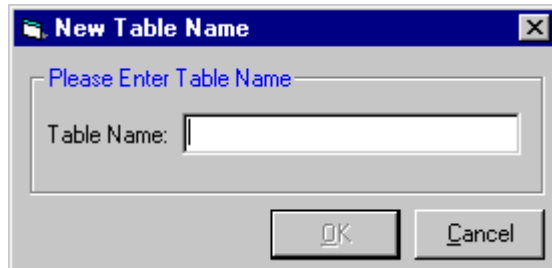
This writes the table to the UDD.

To write more than one table to the UDD, select the check box beside the tables that you want to write and then click the **Write Chosen** button.

Step 13: Create Data, Key and File Information Manually

If you do not have a COBOL FD for a particular file, you can enter the **Data**, index **Keys** and **File** information manually.

Click the **Create...** button on the **COBOL FD Converter** dialog box. The **New Table Name** dialog box is displayed:



Enter the name of the table and click **OK**.

The **Table Maintenance** property sheet is displayed with 'blank' **File**, **Data** and **Keys** property pages. Enter the information as described in Steps 9 to 12.

Step 14: Repeat the Procedure for all your Application's Files

Perform steps 6 to 13, to set up all the tables in your UDD for all your application's files. You can now close the dictionary using either:

- The **Close** button, or



- The icon (close book icon) on the tool bar before finally exiting from the U/SQL Manager.

Next Steps

After completing the loading of your UDD:

- If you wish to modify the UDD refer to the next section, [Modifying a UDD](#), otherwise;
- For Single-tier U/SQL ensure that the [U/SQL Client directives](#) are set up.
- For Multiple-tier U/SQL ensure that the U/SQL Server and Client directives are set up.
- At this point, you are ready to use U/SQL with your ODBC-enabled products on your client platform.

Modifying a UDD

Modifying a UDD

This section describes how you can view and modify an existing dictionary using the U/SQL Manager.

It also describes how you can export your UDD and re-import. You may want to perform this export/re-import process on your UDD:

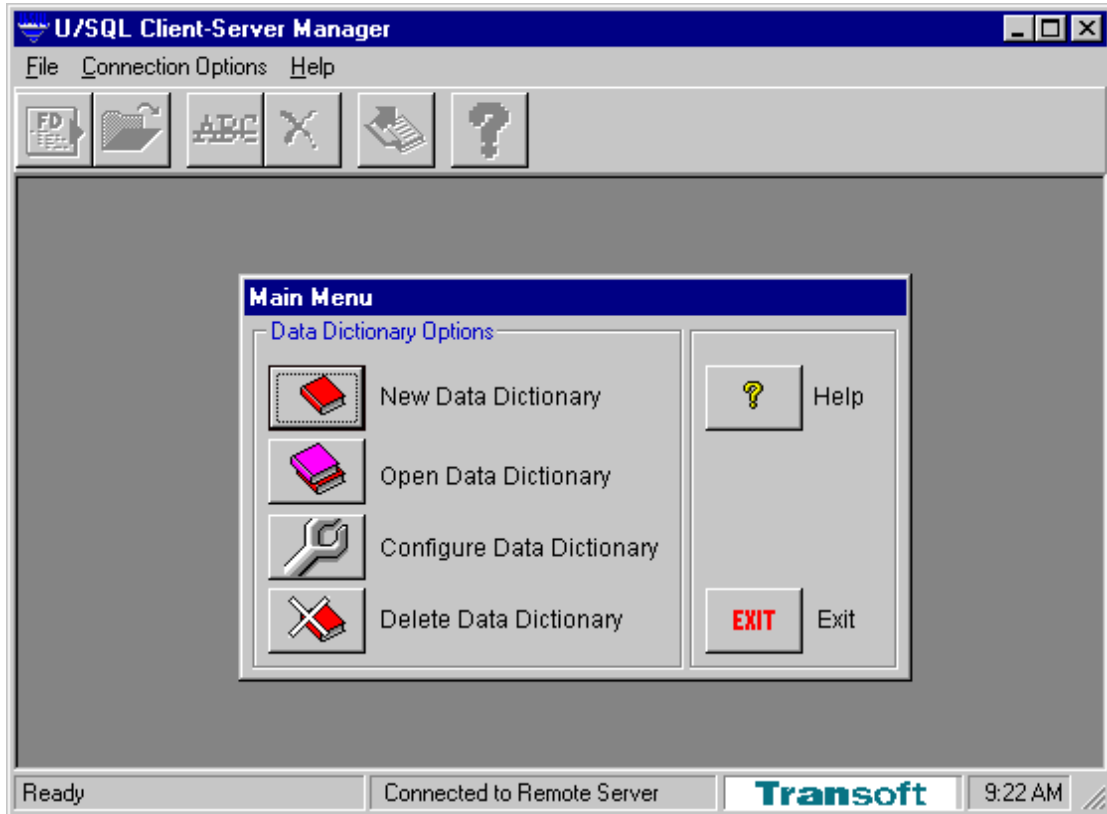
- to rev-up the UDD to the latest revision to obtain the benefit of possibly improved performance, particularly for large dictionaries containing many tables and columns, or to be able to specify more than 8 component parts in any index, which was introduced in revision 2.66, or
- to remove any inconsistencies or corruption in the UDD.

Open an Existing Data Dictionary

Windows

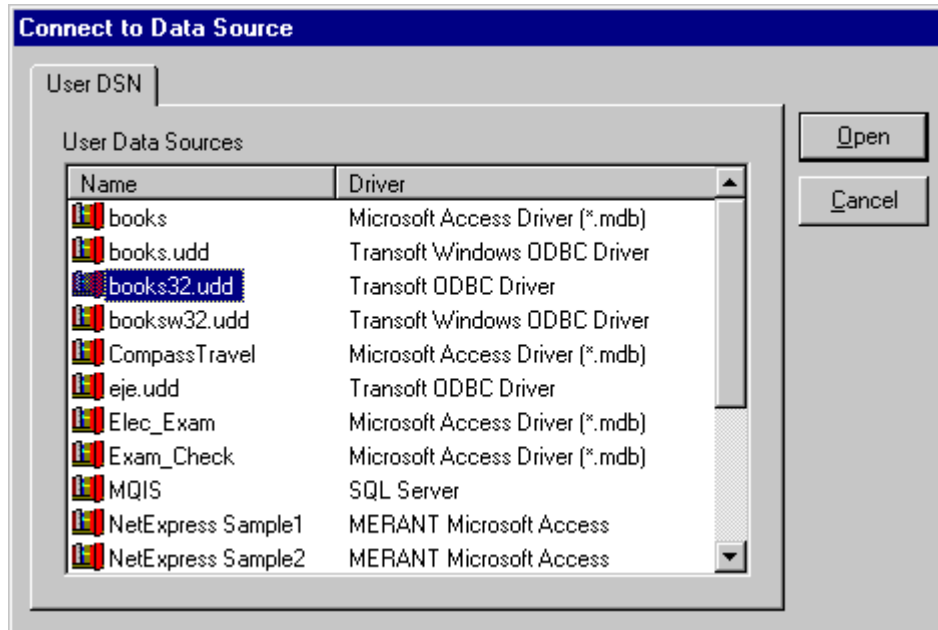
After you have set up a UDD, you can view, enhance or modify it at any time using the U/SQL Manager.

Double-click on the U/SQL Manager icon in the U/SQL program group. The Main Menu is displayed:

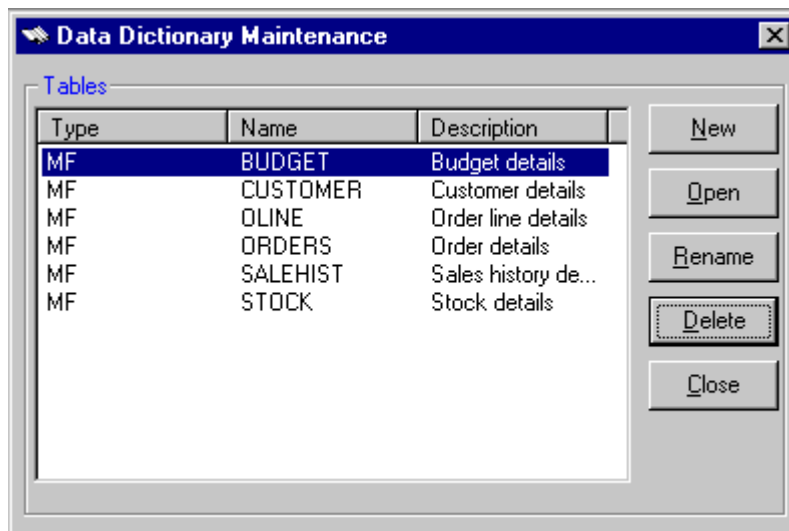


For Multiple-tier installations select the **Remote Server** command from the **Connection Options** menu. For Single-tier installations select the **Local Server** command from the **Connection Options** menu.

Either, click the **Open Data Dictionary** button from the Main Menu, or select the **Open...** command from the **File** menu. The is **Connect to Data Source** dialog box is displayed:



Select an existing Data Dictionary from the list and click **Open** to display the list of tables:

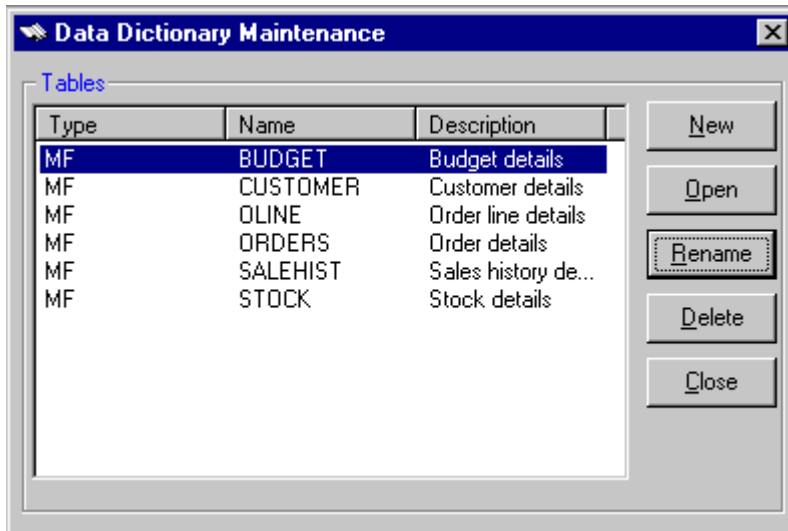


This dialog box allows you to perform the following functions:

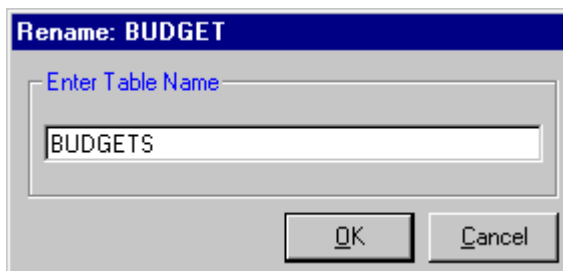
- [Rename an Existing Table](#)
- [Delete an Existing Table](#)
- [View or Modify an Existing Table.](#)

Rename an Existing Table

To rename an existing table, click the table name, for example, BUDGET, and then click **Rename**:



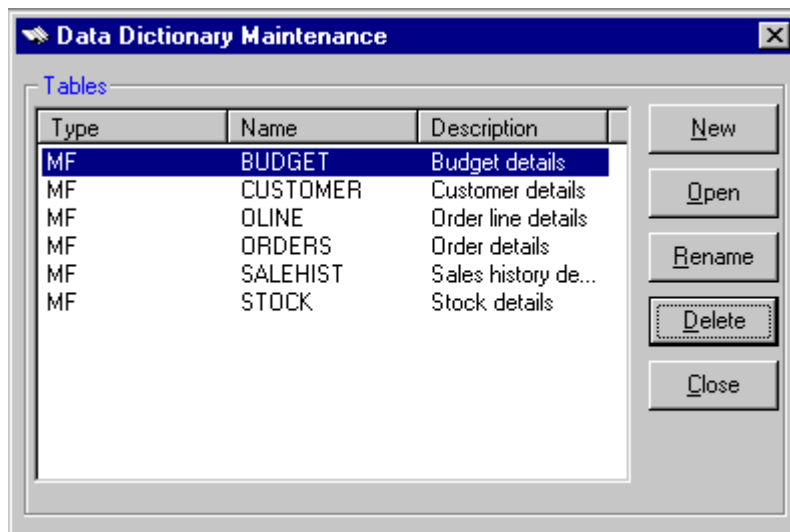
The **Rename** dialog box is displayed:



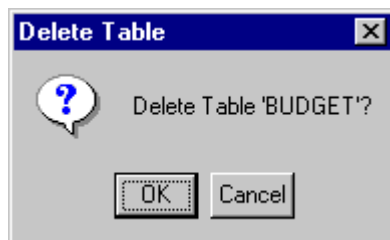
Enter a new name for the table in the **Enter Table Name** field, then click **OK**.

Delete an Existing Table

To delete an existing table, click the table name, for example, BUDGET, and then click **Delete**:



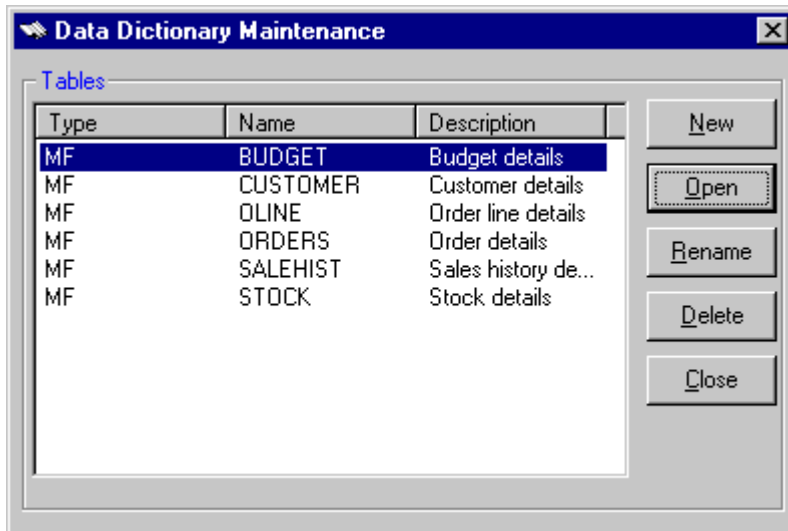
The **Delete Table** dialog box is displayed:



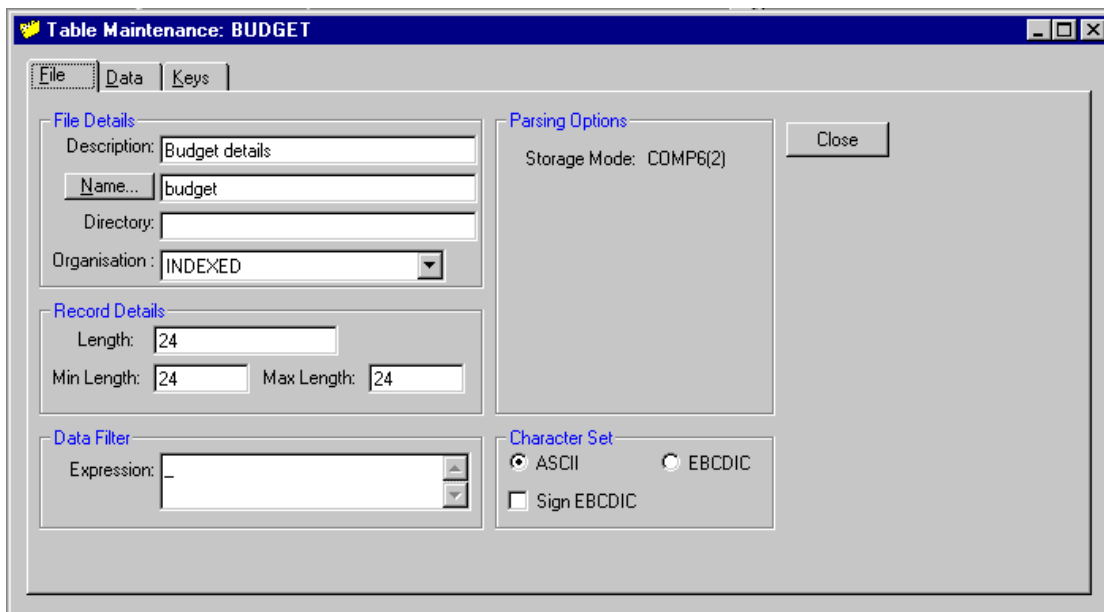
To confirm that you want to delete the table, click **OK**.

View or Modify an Existing Table

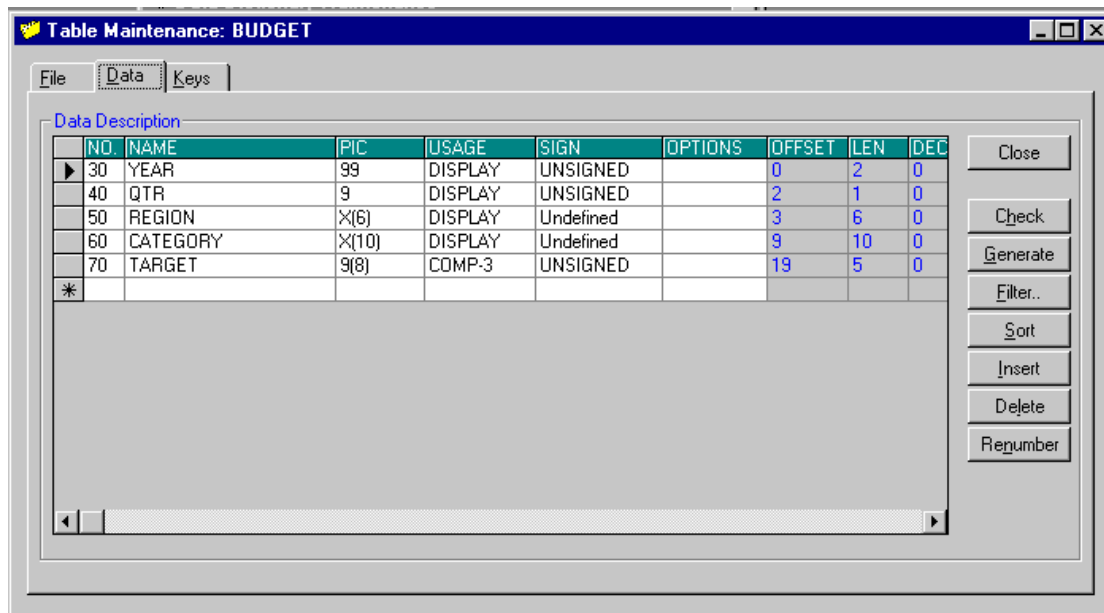
To view or modify a table, highlight the Table and open it by either double-clicking on it or by clicking the **Open** button:



The Table details are displayed:



You can now view or modify any of the **Keys** or **File** information as described previously. However, for the Data information there is only one 'view' of it, as follows:



This Data Description layout is a combination of the two 'views' presented to you by COBOL FD Converter. See the section [Step 9: Manipulating the Data](#). The columns are fully described in this section, but are summarized briefly below:

- **NO** is the same as **SEQ**. You can change the sequence number of the fields and then use the **Sort** button to re-sequence the field names. You can also use the **Renumber** button to renumber the sequence numbers, in steps of 10.
- **NAME**. The field (column) names must be SQL compliant. For assistance, you can use the Check, Generate and Filter... facilities.
- **PIC**, **USAGE** and **SIGN**. These denote the COBOL data type of each field. Note that, if you click on **USAGE** or **SIGN**, a button appears and when clicking on it, a menu appears of the available options.
- **OPTIONS**. This column can contain a combination of "E", "N", "D" and "O" indicating that an **Expression**, **Null Values**, **Date** or **Occurs** has been defined for any field in the record. In addition, it can contain "V" indicating that the field is a **Virtual column**. See the section below [Adding a New Column \(Field\)](#). If you wish to declare any of these options for a particular field, click the **OPTIONS** entry for that field and a button appears. Click the button to display a sub-menu, which contains the following commands:

Expression...

Virtual...

Null Values...

Date...

Occurs...

- **OFFSET**, **LEN** and **DEC**. These define the byte offset of the field, its length, and the number of decimal places, respectively.

Adding a New Column (Field)

Only after you have used the **COBOL FD Converter** to create an initial view of a table (record), can you add additional columns (fields). These can be 'real' columns with the appropriate COBOL data types and byte offsets (for example, because the FD used was an 'old' one and the actual record now has further

fields). They can also be 'virtual' columns that have a column expression added that determines its value.

To add a new column or field to the table, click on the existing column name above which the new column is to be inserted, for example, **REGION**, and click the **Insert** button and the new column is created with a default name of **FIELD1**:

| Data Description | | | | | | | | | |
|------------------|----------|-------|-----------|-----------|---------|--------|-----|-----|--|
| NO. | NAME | PIC | USAGE | SIGN | OPTIONS | OFFSET | LEN | DEC | |
| 30 | YEAR | 99 | DISPLAY | UNSIGNED | | 0 | 2 | 0 | |
| 40 | QTR | 9 | DISPLAY | UNSIGNED | | 2 | 1 | 0 | |
| 50 | REGION | X(6) | DISPLAY | Undefined | | 3 | 6 | 0 | |
| 55 | FIELD1 | - | Undefined | Undefined | | 0 | 0 | 0 | |
| 60 | CATEGORY | X(10) | DISPLAY | Undefined | | 9 | 10 | 0 | |
| 70 | TARGET | 9(8) | COMP-3 | UNSIGNED | | 19 | 5 | 0 | |
| * | | | | | | | | | |

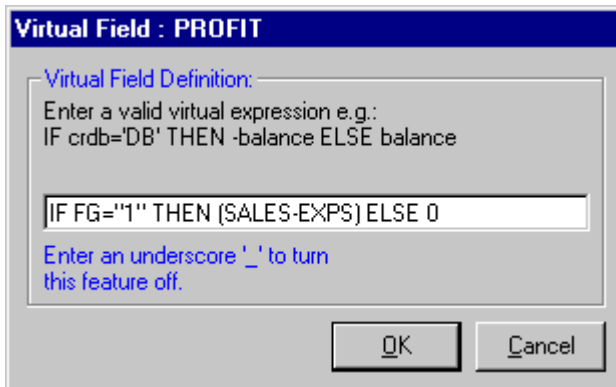
You click the **NAME** to change it and tab across each item in turn entering the appropriate values.

A new column or field can be added to the end of the table by entering the details in the 'blank' row automatically provided.

Note: Adding new fields or columns may affect the record length, which would require changing in the File details. Refer to the section [Step 11: File Details](#).

Virtual Field or Column

If you require a column that does not physically exist, but whose value is determined by an expression, create a new column as above and then click the **OPTIONS** entry for the new column. A button appears. Click the button, and select the **Virtual...** command from the menu. The **Virtual Field** dialog box is displayed:



You can add a column **Expression** in exactly the same way as adding one for a 'real' column; then click **OK**. Refer to the section [Expressions on Columns and Virtual Columns](#).

After adding the new columns to the table, click the **Close** button, and the details are written back to the dictionary.

Deleting a Column (Field)

To remove a column (field) entry, click on its **NAME** entry and then click the **Delete** button, and it is deleted from the view.

Exporting and Importing your UDD

You may want to perform this export/re-import process on your UDD:

- to rev-up the UDD to the latest revision to obtain the benefit of possibly improved performance, particularly for large dictionaries containing many tables and columns, or to be able to specify more than 8 components in any index, which was introduced in revision 2.66, or
- to remove any inconsistencies or corruption in the UDD.

The export/re-import process can be achieved either using the client based Interactive U/SQL utility, [Win U/SQLi](#), or for Multiple-tier U/SQL Adapters with a UNIX server via a script **cobol_rebuild.sh**.

Multiple-tier UNIX

For Multiple-tier U/SQL with a UNIX server your dictionary can be exported and then re-imported automatically.

The script **cobol_rebuild.sh**, which is contained in the **bin** directory below the U/SQL Server software installation directory, for example, **/usr/usqlcs/bin** is responsible for the following:

- backing up your specified UDD, for example, copying the dictionary **yourdict.udd** into the sub-directory backup as **yourdict.udd.18Apr1998**. Note that the date is added.
- exporting the Universal File Dictionary (UFD) information to a text file, in this case **yourdict.txt**.
- creating a new dictionary, in this case **yourdict.udd**.
- re-importing the textual UFD description, **yourdict.txt**.
- updating the dictionary.
- producing a list of tables which have failed to update due to inconsistent information within them.
- producing a file of successful and unsuccessful updates, in this case **yourdict.log**.

Here is an example:

```
$ cobol_rebuild.sh yourdict.udd
Backing up data dictionary yourdict.udd to
    backup/yourdict.udd.18Apr1998...
Exporting UFD tables...
Creating blank data dictionary...
Importing UFD tables...
Updating data dictionary...
Finished.
```

It is also possible to export your UDD to a UFD textual file, using the script **cobol_export.sh**, for example:

```
cobol_export.sh yourdict.udd yourdict.txt
```

You may want to amend the text file, for example, to remove some tables, before using use the script, **cobol_import.sh**, to re-import your UDD. But first you must create a new empty UDD, say, **newdict.udd**, using the Interactive U/SQL Utility, [usqli](#), with the **-c** switch. For example:

```
usqli -c newdict.udd
cobol_import.sh newdict.udd yourdict.txt
```

Then you must then invoke **usqli** with the **-u** switch to update the internals of the dictionary with the information from the imported text file. For example:

```
usqli -u newdict.udd
```

Finally, you are now able to use this **newdict.udd**.

Note: *Great care must be exercised in amending the exported text file. Refer to the next section, [Manually Creating a COBOL UDD](#) to understand the contents of this text file.*

Next Steps

After completing the loading of the UDD:

- For Single-tier U/SQL ensure that the [U/SQL Client directives](#) are set up.
- For Multiple-tier U/SQL ensure that the U/SQL Server and Client directives are set up.
- At this point, you are ready to use U/SQL with your ODBC-enabled products on your client platform.

U/SQL Manager Error Messages

When you use the U/SQL Manager there are various messages that may be displayed. These messages fall into the following categories:

- [ODBC error messages](#)
- [UDD error messages](#)
- [Table and column names error messages](#)
- [COBOL FD Converter error messages.](#)

The following tables detail these error messages and the possible reasons.

ODBC Error Messages

| ODBC Error Messages | Possible Problem |
|--|--|
| "Invalid Port Number: " "Invalid Host Name: " | Multiple-tier only. U/SQL Server not started Incorrect ODBC.INI directive. |

UDD Error Messages

| UDD Error Messages | Possible Problem |
|--|---|
| "Cannot read this INI file in your Windows directory." | ODBC.INI cannot be read Cannot find ODBC.INI file ODBC.INI is read only |
| "Could not create Data Dictionary." | Creating UDD Multiple-tier. Cannot connect to U/SQL Server or it is not running. |
| "An entry for this Data Dictionary already exists in your ODBC.INI file." | Creating a UDD Name already exists. If you continue, the existing entry will be overwritten. |
| "The U/SQL Adapters Manager does not support any of the Data Sources that your U/SQL Server supports." | Opening UDD Mismatch between contents of UDD and the U/SQL Data Source Driver you are running. |
| "The local USQL.MDB database appears to be corrupt. Will attempt to | Opening UDD Your PC has crashed previously. The Manager |

| | |
|--|--|
| repair and try again." | will attempt to repair automatically the local USQL.MDB database. |
| "The attempt to repair the local USQL.MDB database failed. Aborting." | Opening UDD The Manager could not repair USQL.MDB. Copy USQLBAK.MDB to USQL.MDB. These files are in the LOCALDB sub-directory of the U/SQL Manager installation directory. No loss of UDD data is incurred. |
| "Error while opening the local information database." "Error while closing the local information database." | Opening UDD Local USQL.MDB database corrupt due to, say, PC crash. Copy USQLBAK.MDB to USQL.MDB. These files are in the LOCALDB sub-directory of the U/SQL Manager installation directory. No loss of UDD data is incurred. |
| "The following table has a duplicate definition:" | Opening UDD UDD possibly corrupt. Retrieve last valid backup. |
| "Unable to update Data Dictionary for table:" | Updating UDD Invalid data in table definition. Check field names and key fields are valid. |
| "Could not add new field to table:" "Could not add new key to table: " | Insert Field or Key - Updating UDD Table Corrupt UDD. Retrieve last valid backup. Multiple-tier. Cannot connect to U/SQL Server or it is not running. |
| "This Table Name cannot be used in the Data Dictionary because:" "This Field Name cannot be used in the Data Dictionary because:" | Non-compliant names See Table and Column Names Error Messages below. |
| "Some of the Field names will be rejected by the Data Dictionary." | Writing Table to UDD Illegal field names. Use Check button to view illegal names and alter them. |
| "Group fields cannot be written to the Data Dictionary." | Elementary fields Only elementary fields can be written to the UDD. |
| "The Record Number field must be written to the Data Dictionary for Relative files." | Record Number The Record Number must be included for Relative files (tables). |
| "Could not delete Data Dictionary" | Deleting UDD The UDD does not exist Multiple-tier. Cannot connect to U/SQL Server |

| | |
|--|-----------------------|
| | or it is not running. |
|--|-----------------------|

Table and column names error messages

| Table and column names Error Messages | Possible Problem |
|--|--|
| "The name is too long." "The name is an SQL keyword." "The first character of the name is illegal." "The name contains an illegal character." "The name is blank." | Refer to the section Structure View for SQL naming compliance. |

COBOL FD Converter Error Messages

| COBOL FD Converter Error Messages | Possible Problem |
|--|---|
| "No such source file." | COBOL source file cannot be found. Invalid path or name. |
| "No such copy file." | Copyfile referenced in FD cannot be found. Check file exists or check copyfile path. |
| "Too many nested COPY statements." | The limit is 10 levels of nesting. |
| "Tried to find a non-existent field." "Key field defined does not exist." "No parts found for a split Key." | Invalid COBOL FD code. Check code with your COBOL compiler. |
| "Cannot find an FD for this file. Try changing the Source File Options that are available from the Options button on the FD Converter." "No fields found in this FD." | Try a different file format option, for example, Fixed or Free and click Convert again. If message persists check code with your COBOL compiler. |

| | |
|---|--|
| <p>Try changing the Source File Options that are available from the Options button on the FD Converter and then re-Converting the Source File."</p> <p>"Invalid/unprocessable PICTURE clause."</p> <p>"Can't find any SELECT or FD statements in this file. Try changing the Source File Options that are available from the Options button on the FD Converter."</p> <p>"Cannot convert Field definitions. Try changing the Source File Options that are available from the Options button on the FD Converter and then re-Converting the Source File."</p> <p>"Unable to determine source file format."</p> <p>"Cannot convert this file. The file may not be a valid Cobol Source File, or it may be corrupt."</p> <p>"Invalid field definition encountered. Try changing the Source File Options that are available from the Options button on the FD Converter."</p> | |
| <p>"Too many words in a single COBOL scope (maximum 250). It may be either a very large comment in the COBOL Program or you have chosen the wrong Source File Format. Either delete the comment or try changing Source File Formats."</p> | <p>The Cobol FD Converter has a limit of 250 words in a single scope. Edit FD source manually.</p> |
| <p>" Please check the Key Definitions for this table. Either the Key fields do not exist, or the FD Converter was unable to determine them for you automatically."</p> | <p>Illegal or unknown Key fields have been defined.</p> |
| <p>"Please enter a file name."</p> | <p>No file name has been entered.</p> |

"An FD table called " " already exists. Please supply a new table name."

Rename to a unique name.

Manually Creating a COBOL UDD

Manually Creating a COBOL UDD

This section describes how the dictionary can be built by detailing the layout of the application's data files in a text file from which the UDD is created. Use this approach if you want to generate a dictionary from your COBOL application automatically (say, from data stored in a case tool) or where the data uses COBOL data types in, say, C-ISAM as the data source, but where there are no COBOL FDs.

This is applicable to both Single and Multiple-tier U/SQL.

Introduction

You define the contents of your COBOL data dictionary by specifying three SQL tables, **cobol_table**, **cobol_field** and **cobol_index**, that hold details of your files, their fields and indexes in a text file.

You then create an empty dictionary and import into it the contents of this text file creating the Universal File Dictionary (UFD), that is the details of the physical files. The Universal Data Dictionary (UDD) is automatically created from the UFD information. The UDD contains details of the 'relational' tables and their column names that are visible to ODBC-enabled products.

Note: *The layout of the text file is the same as the textural form of an exported dictionary that was originally created using the U/SQL Manager.*

Defining a COBOL Data Dictionary

Create a Text File

The data dictionary is specified by creating a text file, for example, **dictionaryname.ufd**. It consists of three sections, each of which relates to a table in the UFD; **cobol_table**, **cobol_field** and **cobol_index**. Each table is specified as an entity, and a table type can be repeated as any number of entities in the text file. A table entity consists of the following:

```
table_name1(colname_1,colname_2,...)
-----
R1C1  R1C2  R1C3
R2C1  R2C2  R2C3
# Comment.
R3C1  R3C2  R3C3
*
```

- The first line contains the name of the table followed by the names of the columns delimited by commas and enclosed in parentheses.
- The next line contains a template which describes the position and maximum length of the data for each column. The format of this template is one or more groups of dashes, where the position and length of each group specifies the location and maximum length of the following data. The amount of white space between the groups of dashes is not significant.
- There then follows one or more rows of data to be inserted into the UFD table. The row data must match the positions specified in the template.

'#' in the first column of a line marks the beginning of a comment.

'\' in the last column of a line signifies it continues on the next line.

Separate tables are delimited by an asterisk '*' in the first column on a line.

Comma Delimited UFD File

Some of the lines in the UFD text file can become very long, particularly if many index key components are required, making it difficult to manipulate. To overcome this a comma delimited form of the UFD file can be specified.

This is achieved by removing the template and specifying each field item for each row of data separated by commas. String values must be enclosed in double quotes, "".

The special characters still apply; '#', comments character, '\' continuation character and '*' end of table character.

An example table:

```
table_name1(colname_1,colname_2,...)
R1C1,"R1C2",R1C3
12345,"Joe Smith",5678.76
# Comment.
R3C1,"R3C2",R3C3
*
```

You will need to perform the following steps to create tables to specify your COBOL data files and their structure:

- [Step 1: Define Datatables](#)
- [Step 2: Define Fields \(Columns\)](#)

- [Step 3: Define the Indexes.](#)

Step 1: Define Datatables

The UFD table **cobol_table** defines all the table names that are visible to an ODBC-enabled product and their equivalent physical file names. The following example shows the contents of this UFD table:

```

cobol_table(tablename, filename, pathid, filedesc, reclen, maxreclen,
minreclen, nfields, nindices, filetype, compflags, arrays,
recordexpr,
revision, comptype, recexpr)
-----
STOCK      stock      0 Stock File      82 82 82 7 1 41 0 0 _ 2 1 _
CUSTOMER   customer 0 Customer File 40 40 40 3 1 41 0 0 _ 2 1 _
*
```

where:

cobol_table is the internal UFD table name

and the columns are:

| Column | Data Type | Description |
|-----------|-----------|--|
| tablename | char (32) | The SQL table name, that is, the name that is visible to ODBC-enabled products. This name must be SQL compliant and satisfy the following conditions. It must be: <ul style="list-style-type: none"> • in UPPERCASE • within 18 characters • unique • not an SQL keyword • not zero length • made up of 0-9, A-Z, _, with the first character alphabetic and with no blanks. |
| filename | char(128) | The physical diskfile name. On UNIX, the filename is case sensitive. |
| pathid | smallint | The directory path number to the physical file. It is recommended that this is set to zero and that the SearchList directive is used to determine the location of all data files. |
| filedesc | char(40) | A description of the tablename. |
| reclen | integer | If fixed length record, the record length. If not fixed length set to zero. |
| maxreclen | integer | If variable length record, the maximum record length. If fixed length set to zero. |
| minreclen | integer | If variable length record, the minimum record length. If fixed length set to zero. |

| | | |
|------------|-----------|--|
| nfields | smallint | The number of fields in the record. Maximum 512. |
| nindices | smallint | The number of indexes defined on the table. |
| filetype | smallint | The type of the file, which can have the following values: 41 - Indexed 42 - Relative 43 - Sequential 44 - Line Sequential 45 - Variable Sequential 46 - Variable Relative 47 - Variable Indexed |
| compflags | integer | Bit field containing compiler directives in force on the data file. Acucobol Flags 1 - D1 2 - Dm 4 - D5 8 - D6 16 - Di 32 - Ci 64 - Dca 4096 - Dci 8192 - Df Micro Focus Flags 256 - ODOSLIDE 1024 - EBCDIC 2048 - EBCDIC SIGN |
| arrays | smallint | Set to 1 if the table contains one or more arrays, otherwise 0 (zero). |
| recordexpr | char (64) | No longer used. |
| revision | smallint | The revision of the UDD. Used internally. |
| comptype | smallint | No longer used. |
| recexpr | char(254) | Conditional test to establish if a record belongs to table. (This is an expression which can be a maximum of 254 characters and as a minimum there must be an '_' entry). See the section Multiple Record Type Handling below. |

Multiple Record Type Handling

If there are multiple records defined in the same physical file, each record type can be described as a separate logical table. You need to know how each record type can be uniquely distinguished from the others. This is usually determined by

one or more fields having particular values. You must decide on the different names you wish to call each logical table and establish the differentiating values or expressions. For further information, see the section [Dealing with Multiple 01 Records](#).

An example expression is:

```
REC_TYPE="P"
```


Step 2: Define Fields (Columns)

The UFD table `cobol_field` defines all the field (column) names that are visible to ODBC-enabled product and their physical attributes. The following example shows the contents of this UFD table.

```
cobol_field (tablename, fldname, cobolname, flddesc, sequenceno, type,
usage, signed, len, offset, ndec, picture, levelno, numdigits,
dateformat,
fieldflags, arraylevname, arraylevel, nullvalue, colassignexpr,
scaling)
```

```
-----
STOCK STKNUM      _ _ 30 2 10 0 6 0 0 X(6) 5 0 _ 1 _ 0 _ _ NULL
STOCK TOCK_TITLE _ _ 40 2 10 0 30 6 0 X(30) 3 0 _ 1 _ 0 _ _ NULL
STOCK PUBLISHER  _ _ 50 2 10 0 10 36 0 X(10) 3 0 _ 1 _ 0 _ _ NULL
*
```

where:

`cobol_field` is the internal UFD table

and the columns are:

| Column | Data Type | Description |
|------------|-----------|---|
| tablename | char(32) | The SQL table name, that is, the name that is visible to ODBC-enabled products. It is the same name as in <code>cobol_table</code> . |
| Fldname | char(32) | The field (column) name that is visible to ODBC-enabled products. This name must be SQL compliant and satisfy the following conditions. It must be: <ul style="list-style-type: none"> usually within the SQL standard of 18 characters, but can be a maximum of 30 characters. unique not an SQL keyword not zero length made up of 0-9, A-Z, a-z, _, with the first character alphabetic and with no blanks. |
| cobolname | char(32) | The COBOL field name. |
| flddesc | char(40) | A field description. |
| sequenceno | smallint | The sequence number of the field. The first field is set to 10 and then increments in steps of 10 (in case new fields need to be inserted). |
| type | smallint | The data type, which can be: 1 - Alpha |

| | | |
|------------|----------|---|
| | | <p>2 - Alphanumeric 3 - Numeric</p> |
| usage | smallint | <p>The usage type of the field, which can contain the following values:</p> <p>10 - DISPLAY 11 - BINARY 12 - COMP 13 - COMP-1 14 - COMP-2 15 - COMP-4 16 - COMP-5 17 - COMP-6 BCD 18 - COMP-X 19 - COMP-N 20 - PACKED-DECIMAL 21 - COMP-3 22 - INDEX 23 - COMP-0 24 - PIC-E 25 - COMP-6 BINARY 26 - FLOAT 27 - DOUBLE 28 - Acu COMP-1 29 - Acu COMP-2 36 - Packed integer</p> |
| signed | smallint | <p>Sign type, which can have the following values:</p> <p>30 - Leading 31 - Trailing 32 - Leading separate 33 - Trailing separate 34 - Unsigned 35 - Signed</p> |
| len | integer | <p>The length in bytes of the field (column). The maximum SQL string data type is 254 bytes. Split fields (columns) greater than this maximum into multiple columns.</p> |
| offset | integer | <p>The offset within the record from byte zero.</p> |
| ndec | smallint | <p>The number of decimal places (if applicable).</p> |
| picture | char(30) | <p>The COBOL picture string for the field.</p> |
| levelno | smallint | <p>The level number of the field, for example, 01, 03 and so on.</p> |
| numdigits | smallint | <p>The number of digits in the field.</p> |
| dateformat | char(30) | <p>The date format if the field is to be defined as a date. This can hold either a Date Format</p> |

| | | |
|---------------|-----------|---|
| | | String or a Julian Date Offset , where 1 is 1 January 0001. For example, a data format string could be MMDDYYYY and a Julian date offset for 1 January 1968 is 718432. Leave blank if not a date. See the section SQL Date Formats . |
| fieldflags | integer | <p>A bit field that contains attributes for the field:</p> <ul style="list-style-type: none"> 1 - field used flag (if not set, entry in cobol_field table will not be used). 2 - documentary - used by the U/SQL Manager to identify a group field. 4 - documentary - used to identify an entry for an array header. 8 - field used as an array subscript. 16 - field is a member of an array element. 32 - documentary - used to identify individual array field by U/SQL Manager. 64 - field is used to return record number. 128 - documentary - identify field that controls OCCURS DEPENDING ON. 256 - virtual column. 512 - documentary - identifies subscript for parallel array processing. 1024 - field has expression to define logical null. 2048 - Hidden field. Refer to the section Hidden Fields, below. <p>The bits that are marked as 'documentary' are used by the U/SQL Manager when processing the cobol_field table. It is desirable, but not essential, to set these bits when creating the UDD from a UFD text file.</p> <p>This field must never contain zero, as a minimum it should be 1.</p> |
| Arraylevname | char(18) | The array level name of field if part of the array. |
| arraylevel | smallint | The array level number of field if part of the array. |
| nullvalue | char(64) | An expression that gives the NULL value for a column. |
| Colassignexpr | char(254) | An expression that changes the value of the column. Refer to the section Expressions on Columns and Virtual Columns . |
| Scaling | smallint | Reserved for future use. |

Hidden Fields

Setting bit 12 of **fieldflags** (decimal 2048) in the **cobol_field** table results in the column being hidden from the user. The column can still be queried and, if applicable, can be used by the query planner for optimum performance.

Hidden fields cannot currently be set by the U/SQL Manager.

Note: *As well as setting bit 12 one other bit needs to be set to produce a valid configuration. Hence, **fieldflags** usually have a value of 2049, that is bit 12 and bit 1 are set.*

Data Redundancy in Alternate Keys

Assuming the following example:

```
01 MAIN-REC.
   05 KEY1.
       10 CUST1          PIC X(8) .
       10 REGION1       PIC 99.
   05 KEY2.
       10 REGION2       PIC 99.
       10 CUST2         PIC X(8) .
   05 MAIN-DATA.
```

There is a duplication of data for the fields for **cust** code, and **region**.

If the fields of the alternate key (REGION2, and CUST2) are hidden (see above section) and the user chooses to select using the visible region field (REGION1) then the WHERE clause will not reference the second region field and hence will not allow the U/SQL Server engine to use the alternate index (the same applies for an ORDER BY clause).

However if a record expression (refer to the section [Expression Handling Syntax](#)) is applied to the table as follows:

```
CUST2 = CUST1 and REGION2=REGION1
```

then this will be propagated into the WHERE clause on each SELECT and the syllogistic reasoning of the query planner will transform

```
REGION1='XXX' AND REGION2=REGION1
```

into

```
REGION1='XXX' and REGION2=REGION1 and REGION2='XXX'
```

and hence use the alternate index for this access.

This could be extended for any number of occurrences, for instance:

```
CUST2 = CUST1 and REGION2=REGION1 and CUST3 = CUST1 and
REGION3=REGION1
```

The syllogistic reasoning is undertaken for both **WHERE** and **ORDER BY** clauses.

Step 3: Define the Indexes

The UFD table **cobol_index** defines all the indexes associated with each **tablename** declared in the table, **cobol_table**. The following example shows the contents of the **cobol_index** UFD table.

```
cobol_index (tablename, idxname, idxno, duplicates, fld1,
fld2, fld3, ..... fld64)
```

```
-----
STOCK      STOCKK0      0  101 STKNUM      -    -    -
CUSTOMER  CUSTOMERK0  0  101 CUST_CODE  -    -    -
*
```

where:

cobol_index is the internal UFD table

and the columns are:

| Column | Data Type | Description |
|------------|-----------|---|
| tablename | char(32) | The SQL table name that this index is for. It is the same name as in cobol_table . |
| Idxname | char(32) | The index name. |
| indno | smallint | The sequence number of the index within the file. |
| duplicates | smallint | 101 - index is unique. 100 - duplicate indexes allowed. |
| fld1 | char(32) | The first field in the key. |
| fld64 | | Up to 64 fields are permissible in the key. You only need specify, in the template, as many index fields as are used. |

Creating the UDD from the UFD Text File

You create your UDD, either using:

- The Windows Interactive U/SQL utility, [Win U/SQLi](#), on a client PC, or
- On the server for UNIX platforms, using the Interactive U/SQL utility, [usqli](#).

Single-tier or Multiple-tier

When you have created the text file defining the UFD structure of your COBOL files, for example **demo.ufd**, you use the U/SQL Client software version of the Interactive U/SQL utility, **Win U/SQLi**, to create your UDD:

1. For Multiple-tier U/SQL ensure that the U/SQL Server is running on your UNIX or Windows NT Server machine.
2. Invoke **Win U/SQLi**, by double-clicking the **Win USQLi** icon in the U/SQL Client program group, on the client PC where you have created the UFD text file, for example, **demo.ufd**.
3. Create a new UDD, for example, **demo.udd**, to which the UFD text file will be imported, by either clicking on the **Create a new UDD** icon from the toolbar, or selecting **New UDD** from the **File** menu. Select where you want the UDD to be created, that is **Local** for Single-tier or **Remote** for Multiple-tier.

You will then set up the [ODBC.INI directives](#).

4. Click the **Import Tables** icon or select **Import** from the **Table** menu. Select the UFD text file you have created, in this example, **demo.ufd**.
5. An **Import Tables** dialog box is displayed, showing each table as it is imported. Each successfully imported table has 'Imported' displayed to the left of the table name.
6. After all the tables have been imported, click **OK**.
7. Click the **Update selected tables** icon, or select **Update** from the **Tables** menu. Select the appropriate data source from the dialog, and click the **Update** button. This creates the UDD tables from the UFD tables.
8. The message, *All tables successfully updated*, is displayed.

There are conditions where you do not get this message, for example, due to a missing key component in an index. Interrogate the log file to obtain which table has the problem and what it is. Refer to the section [How to Query the Log File](#).

9. Click **OK**.
10. You can check that the new UDD is correct by performing one or more queries on it using **Win U/SQLi**.

You have now created and populated your UDD.

Note: *Ensure you perform the update step (7), otherwise the UDD is left in an incomplete state and is unusable.*

Multiple-tier with UNIX Server

UNIX

On Multiple-tier UNIX platforms, once you have created the text file containing the entries defining the UFD structure of your COBOL data files, for example, **demo.ufd**, you must go through a three-step process to create the UDD:

1. In the **bin** directory below the base directory where you loaded the U/SQL Server software, for example, **/usr/usqls/bin**, create an empty UDD, for instance, **demo.udd**, by running the Interactive U/SQL utility [usqli](#), with the **-c** switch:

```
cd /usr/usqls/bin
./usqli -c demo.udd
```

Note: *The dictionary must have a '.udd' extension.*

2. Next, you must load your UFD data, **demo.ufd**, into the UDD. This is achieved by running **usqli** again with just the name of the UDD:

```
./usqli demo.udd
```

and then, at the 'U/SQL' prompt:

```
U/SQL> import demo.ufd
U/SQL> quit
```

3. Finally, you must create the UDD, from the UFD information, by running **usqli** with the **-u** switch:

```
./usqli -u demo.udd
```

The message, `All tables successfully updated`, is displayed.

There are conditions where you do not get this message, for example, due to a missing key component in an index. Interrogate the log file to obtain which table has the problem and what it is. Refer to the section [How to Query the Log File](#).

You can update one table at a time:

```
./usqli -u demo.udd AREAS
./usqli -u demo.udd COUNTIES
./usqli -u demo.udd CUSTOMERS
```

The UDD, **demo.udd**, is now ready for use.

To list the tables just loaded into the UDD, run **usqli**:

```
./usqli demo.udd
```

and then, at the 'U/SQL' prompt:

```
U/SQL> select tablename from cobol_table;
.
. [tablename table names listed]
.
U/SQL> quit
```

Note: *Ensure you perform the update step (3), otherwise the UDD is left in an incomplete state and is unusable.*

Amending the Data Dictionary

If any amendments are required then either:

- The existing UDD must be deleted, the relevant changes made to the '.ufd' text file and the steps repeated, in the previous section, to create and load the new UDD, or
- A new data file can have a '.ufd' text file made for it and the steps to import and update the UDD repeated.

Note: *If any table requires amending, it is recommended that you recreate the whole dictionary to ensure a consistent UDD.*

Next Steps

After completing the loading of the UDD:

- For Single-tier U/SQL ensure that the [U/SQL Client directives](#) are set up.
- For Multiple-tier U/SQL ensure that the U/SQL Server and Client directives are set up.
- At this point, you are ready to use U/SQL with your ODBC-enabled products on your client platform.

COBOL Issues

There are various issues that must be considered when using U/SQL with an ACUCOBOL or Micro Focus COBOL data source:

- COBOL PIC 9(16), PIC 9(17) and PIC 9(18) data types
U/SQL data sources that support COBOL data types do not support the full range of values held in PIC 9(16) or greater data types. This restriction includes all computational storage options.
- Variable length records are restricted to read-only. Review NULL mapping of columns in the variable portion of the record.

If your COBOL FDs qualify data names in the specification of any index, then the U/SQL Manager may not correctly identify these key fields. It is extremely important to check that entries for each index contain all the key components in the correct order.

This section discusses the following specific issues:

- [Multi-Byte NULL Expressions for COBOL data types](#)
- [Support of ACUCOBOL Vision 4 data files](#)
- [Micro Focus COBOL Issues](#)
- [Support for Micro Focus COBOL Fileshare under UNIX](#)
- [Support for Micro Focus COBOL Fileshare under Windows.](#)

Multi-Byte NULL Expressions for COBOL data types

In revisions of U/SQL prior to Rev 3.10, a NULL expression required a single byte to be repeated throughout the entire field, for the field to be considered NULL. In U/SQL Rev 3.10 and above the NULL expression has been modified to also allow you to enter a sequence of bytes as a pattern that must be matched in the field for it to be considered NULL. You can enter this pattern in any combination of the three formats previously supported, but with the following extensions:

- When the value is in quotation marks, you can specify more than one character inside the quotes. For example, "WEST!".
- Using the hexadecimal representation of ASCII, this same pattern is x57x45x53x54x21
- In the decimal representation of ASCII, this is 087069083084033

You can use combinations of these three formats. For example:

087"EST"x21: combines the decimal representation of ASCII, printable ASCII and hexadecimal.

The resulting patterns are matched against the actual COBOL field.

If you define the NULL expression as a series of bytes, for example, "EAST" then the pattern EAST must be repeated throughout the field. Therefore, given a NULL expression defined as "EAST", a 9 byte field must contain "EASTEASTE" to be considered NULL.

Support of ACUCOBOL Vision 4 data files

The ACUCOBOL Data Source Driver supports Vision 4 data files.

ACUCOBOL Vision 4 is backwardly compatible with Vision version 2 and 3 file formats. Therefore all U/SQL Rev 3.10 ACUCOBOL products will only be available with a Vision 4 file handler.

Micro Focus COBOL Issues

- The combination of IBMCOMP and SYNCHRONIZED directives is not supported.
- The file handler supports record and key compression.
- Micro Focus only fully support their COBOL on Windows NT Server 4 or Windows 2000. If you use Windows NT Server 3.51, then do not logout of the NT console.
- You must have a Micro Focus COBOL runtime to operate Multiple-tier U/SQL.
- Under Multiple-tier UNIX platforms, to ensure locking works consistently, set up the correct environment variables when using Micro Focus COBOL V3.0 or V3.1.

Support for Micro Focus COBOL Fileshare under UNIX

Note: *The information below is based upon U/SQL accessing Object COBOL Developer's Suite 4.1.xx and Server Express v1.xx and above. This is not a definitive guide and reflects usage which is consistent with runtime versions available at the time of writing (consult your Micro Focus documentation for complete instructions on Fileshare usage and configuration).*

As a pre-requisite you need to ensure **CCITCP** is running. If not change directory into the **\$COBDIR** directory and start, as root, the **ccticp2** process (start this as a background job available to your Fileshare server so that it can be referenced through more than one terminal session).

Set the two Fileshare directives in your **usqlsd.ini** file as follows:

```
[books.udd]
Dictionary=../example/books.udd
Directory=../example
AutoTransProc=Y
```

go into the directory where

File Share=Y the datafiles are and, where appropriate, create a database reference file*: This will tell the Fileshare server which datafiles should be made available for Transaction Processing.

```
vi books.ref
/f budget
/f customer
/f orders
/f oline
/f salehist
/f stock
```

Save this '.ref' file in the directory where the datafiles are and set the Fileshare environment variables:

```
$ export FSSERVER=MyFileShareServerName1
$ export FSCOMMS=CCITCP
```

Transoft U/SQL User Guide

Start Fileshare using the parameters below (refer to your Micro Focus documentation for alternative methods or clarification on command line parameters):

```
$COBDIR/fs -s MyFileShareServerName1 -d books.ref -cm ccitcp -tr f
```

Note: *Enable File Share tracing by pressing F2. This will allow you to see transactions taking place and will give visual confirmation that U/SQL is communicating with your Fileshare server process.*

Once you have started and configured all elements relating to your Fileshare environment start your U/SQL server and access a **usqli** session.

At the U/SQL> prompt switch transaction mode on:

```
U/SQL> transaction mode on
```

You can then test whether Fileshare and U\SQL are communicating by following the examples below:

```
U/SQL> select * from customer where custcode >='C020';
```

```
CUSTCODE  CUST_NAME  CUST_REGION
C021  Tonys Book4  WEST
C022  Tonys Book5  WEST
C023  Tonys Book6  WEST
C024  Tonys Book7  WEST
C025  Tonys Book5  WEST
5 records retrieved
```

```
U/SQL> delete from customer where custcode < 'C023';
```

```
U/SQL> select * from customer where custcode >='C020';
```

```
CUSTCODE  CUST_NAME  CUST_REGION
C023  Tonys Book6  WEST
C024  Tonys Book7  WEST
C025  Tonys Book5  WEST
3 records retrieved
```

```
U/SQL> rollback
```

```
U/SQL> select * from customer where custcode >='C020';
```

```
CUSTCODE  CUST_NAME  CUST_REGION
C021  Tonys Book4  WEST
C022  Tonys Book5  WEST
C023  Tonys Book6  WEST
C024  Tonys Book7  WEST
C025  Tonys Book5  WEST
5 records retrieved
```

```
U/SQL>
```

*** If the data files reside in MORE than one Directory then amend the .ref file like so**

```
/f budget
/f customer01 /af test/customer01
/f orders
/f oline
/f salehist
/f stock
```

The **/f datafile** entry points to the current working directory where the data (and .ref) resides. The use of the alternative file **/af** switch tells the Fileshare server to look for the relevant datafile in the base and alternative directories.

Note: *Fileshare will work with substitution as it is U/SQL that handles this.*

Support for Micro Focus COBOL Fileshare under Windows

Due to potential implementation differences on various Windows platforms refer to your Micro Focus documentation regarding the configuration and use of Fileshare on non-UNIX Operating Systems.

Linking external file handlers

Since U/SQL makes use of the standard Micro Focus EXTFH file handler, the C-ISAM library can be linked into U/SQL in the same way it would be linked into a Micro Focus COBOL runtime system.

The makefile shipped with the U/SQL release will need to have the entry:

```
COBLIB= cobol.a
```

modified to:

```
COBLIB= cobol.a -m ixfile=cixfile -L/usr/lib +lisam
```

where `/usr/lib` is the UNIX directory containing the C-ISAM library **libisam.a**.

This mechanism is described in detail in the *Micro Focus reference manual* under the 'Callable File Handler' section. This facility does not apply when the standard Micro Focus file system is used.

U/SQL requires the directive **MFIsam=N** to be included in the **usqlsd.ini** configuration file.

Setting Up a C-ISAM Data Dictionary

Setting Up a C-ISAM Data Dictionary

This section describes setting up a UDD for C-ISAM based data and specifically the data types for Cognos PowerHouse C-ISAM.

Note: Refer to the [Overview](#) and [Creating a UDD](#) sections for an overview of the dictionary technology and the steps involved in creating a UDD.

Introduction

The U/SQL Server supports a data source driver for C-ISAM covering differing data types. Although C-ISAM, as supplied by Informix, defined a limited number of 'C' data types, many languages and applications make use of C-ISAM as their data source each of which has its own data types, for example, dates are typically held in proprietary formats. Your particular version of C-ISAM will only use a subset of the full range of data types available. These data types are automatically converted by the U/SQL C-ISAM Data Source Driver into ODBC-compliant data types during run-time.

U/SQL provides both read and write capability, via ODBC-enabled SQL based products, to your existing C-ISAM data files, assuming that you are not restricted to read only, either by Licensing or via the U/SQL Server configuration directive.

The first step is to define the contents of your data dictionary. You specify three SQL tables, **cisam_table**, **cisam_field** and **cisam_index**, that hold details of your existing files, their fields and indexes.

All this information is contained in the Universal File Dictionary (UFD). The Universal Data Dictionary (UDD) is automatically created from the UFD. The UDD contains details of the 'relational' tables and their column names that ODBC-enabled products will 'see'.

Defining a C-ISAM Data Dictionary

Create a Text File

The data dictionary is specified by creating a text file, for example, **dictionaryname.ufd**. It consists of three sections, each of which relates to a table in the UFD:

- `cisam_table`
- `cisam_field`
- `cisam_index`.

Refer to the [Creating a UDD](#) section for how the UFD text file is formatted.

The following steps must be performed to create tables to specify your C-ISAM data files and their structure:

- [Step 1: Define the Data Files](#)
- [Step 2: Define the Fields \(Columns\)](#)
- [Step 3: Define the Indexes](#).

Step 1: Define the Data Files

The UFD table **cisam_table** defines all the table names that an ODBC-enabled product will 'see' and their equivalent physical file names. The following example shows the contents of this UFD table:

```
cisam_table(tabname,reclen,data_file,recexpr)
```

```
-----
AREA          24  AREADATA
COUNTY       24  COUNTYDATA
CUSTOMER      216  CUSTMAST
*
```

where:

`cisam_table` is the internal UFD table name

and the columns are:

| Column | Description |
|---------|--|
| tabname | <p>The SQL table name, that is, the name the ODBC-enabled products will 'see'. This name must be SQL compliant and satisfy the following conditions. It must be:</p> <ul style="list-style-type: none"> • in UPPERCASE • within 18 characters • unique • not an SQL keyword • not zero length |

| | |
|-----------|---|
| | <ul style="list-style-type: none"> made up of 0-9, A-Z, _, with the first character alphabetic and with no blanks. |
| reclen | The record length. |
| data_file | The physical diskfile name, without any '.dat' or '.idx' extension. |
| recexpr | Conditional test to establish if a record belongs to the <code>tablename</code> table. (This is an expression which can be a maximum of 250 characters). See the Multiple Record Type Handling section below. |

Note: Two further columns, **nfield** and **nindex**, are automatically generated in **cisam_table**.

Multiple Record Type Handling

The **recexpr** column in the **cisam_table** can contain a condition or expression. Only records that satisfy the condition or expression will be included in the table. The syntax of the conditional expression is exactly the same as that for an SQL **WHERE** clause search-condition (see the [Expression Handling](#) section) and refers to columns in the table.

For example, assume the file **TRANSMAST** contains two record types, **INVOICES** and **PAYMENTS**. They are distinguished by **INVOICES** having the field (column) **TYPE=1** and **PAYMENTS** having **TYPE=2 AND CREDIT="Y"**. You could therefore define three tables called, say, **TRANSACTION**, **INVOICE** and **PAYMENT**, which will be 'seen' to contain all transactions, just invoices and just payments respectively, when viewed by an ODBC-enabled product. The entries in the **cisam_table** are:

```
cisam_table(tabname,reclen,data_file,recexpr)
```

```
-----
```

| | | | |
|-------------|----|-----------|-----------------------|
| TRANSACTION | 70 | TRANSMAST | |
| INVOICE | 70 | TRANSMAST | TYPE=1 |
| PAYMENT | 70 | TRANSMAST | TYPE=2 AND CREDIT="Y" |

*

Step 2: Define the Fields (Columns)

The UFD table **cisam_field** defines all the field (column) names that an ODBC-enabled product will 'see' and their physical attributes. The following example shows the contents of this UFD table:

```
cisam_field(tabname, fldname, type, length, offset, ndec)
```

```
-----
```

| | | | | | |
|----------|-------------|------|----|----|---|
| AREA | AREA_CODE | long | 4 | 0 | 0 |
| AREA | AREA_DESC | char | 20 | 4 | 0 |
| COUNTY | COUNTY_CODE | long | 4 | 0 | 0 |
| COUNTY | COUNTY_DESC | char | 20 | 4 | 0 |
| CUSTOMER | CUST_CODE | long | 4 | 0 | 0 |
| CUSTOMER | CUST_ADDR | char | 80 | 4 | 0 |
| CUSTOMER | CUST_AREA | long | 4 | 84 | 0 |
| CUSTOMER | CUST_REP | long | 4 | 88 | 0 |
| CUSTOMER | CUST_COUNTY | long | 4 | 92 | 0 |


```
CUSTOMER  CUR_BALANCE  decimal 8   96  2
CUSTOMER  BUD_BALANCE  decimal 8  104  2
CUSTOMER  CREDIT_LIMIT long    4   112  0
CUSTOMER  COMMENTS    char   100 116  0
```

*

where:

`cisam_field` is the internal UFD table name

and the columns are:

| Column | Description |
|-----------|---|
| tablename | The SQL table name, that is, the name the ODBC-enabled products will 'see'. It is the same name as in cisam_table . Must be UPPERCASE. |
| fldname | The field (column) name that the ODBC-enabled products will 'see'. This name must be SQL compliant and satisfy the following conditions. It must be: <ul style="list-style-type: none"> • in UPPERCASE • within 18 characters • unique • not an SQL keyword • not zero length • made up of 0-9, A-Z, _, with the first character alphabetic and with no blanks. |
| type | The data type. For full list, see below. |
| length | The length in bytes of the field (column). |
| offset | The offset within the record from byte zero. |
| ndec | The number of decimal places. |

U/SQL 3.10 and above the **cisam_field** table has the following additional three columns:

```
cisam_field (tablename, fldname, type, length, offset, ndec,
fieldflags, colassignexpr, format)
```

The details of the new columns are as follows:

| Column | Data Type | Description |
|------------|-----------|--|
| fieldflags | integer | A bit field that contains attributes for the field: <ul style="list-style-type: none"> • Bit 9 (decimal 256) - Virtual Columns. Refer to the section Virtual columns below. • Bit 12 (decimal 2048) - Hidden Fields. |

| | | |
|---------------|-----------|---|
| | | Refer to the section Hidden Fields below. |
| colassignexpr | char(254) | An expression that changes or defines the value of the column. Refer to the sections Virtual columns and Column assignment expressions below. |
| format | char(30) | This is a string comprised of a combination of key and non-key symbols, which is used to represent dates, times, or timestamps. |

These columns facilitate the following functionality:

Hidden fields

Setting bit 12 (decimal 2048) of **fieldflags** in the **cisam_field** table results in the column being hidden from the user. The column can be used in a column assignment or record expression and, if applicable, can be used by the query planner for optimum performance.

Column assignment expressions

An expression can be placed on any column (field) to change its value depending on the value of other columns in the same row (record). A column expression has three forms:

1. IF <condition> THEN *assignment_expression*
2. [ELSE *assignment_expression*]
3. *assignment_expression*
4. *\$special_assignment_expression*

where:

condition: consists of one or more predicates, combined with the logical operators **AND**, **OR** and **NOT**.

assignment_expression: can be one of the following:

- a column name, for example, AMOUNT
- a string literal or numeric constant, for example, "Y" or 10
- an arbitrary arithmetic expression, for example: ((FLAG2-3)/6) - (AMOUNT*10) + FLAG2

Consider the following example. Assume a data record has the column BALANCE which always holds the amount as a positive value. A further column CREDIT is set to "N" if the BALANCE amount is negative. For an ODBC-enabled product to be able to provide the summation of the amounts, taking into account whether they are debit or credit values, from a number of rows (records), the value BALANCE could be either:

```
IF CREDIT="N" THEN -BALANCE
```

Or

```
IF CREDIT<>'N' THEN BALANCE ELSE -BALANCE
```

The use of the **colassignexpr** column in the **cisam_field** has been extended to allow the following syntax:

```
$special_assignment_expression:
```

- '\$' has been introduced as a special operator to indicate the current field addressed by the entry
- 'special_assignment_expression' is an expression that resolves to a numeric constant.

For example, suppose the field is a record_type and is stored with the values 18, 19 and 20 but the end user knows these as 1, 2 and 3, then the 'perceived' value of the column must be adjusted by subtracting 17. This is achieved by setting the **colassignexpr** to \$17.

Note: *The value 17 is positive and manipulates the perceived value to make the stored value, not the other way round. Hence using the previous example, an expression of \$-17 would manipulate the column to be perceived to contain 35, 36 and 37.*

Given the correct value, the process is reversible, the value is subtracted on a SELECT and added on an INSERT or UPDATE.

However, there are limitations:

- The fields must be numeric. They cannot be alpha-numeric
- Such fields must not be duplicated (redefined) or part of an array.

Note: *Scalar functions cannot be used in the column expression (**colassignexpr**) columns. To add an assignment expression to a column, set the **colassignexpr** field in **cisam_field** to be the required expression.*

Virtual Columns

A virtual column is not a physical field that exists in the data record. A virtual column can be defined in any table, where the value is determined solely from an expression. To add a virtual column, set bit 9 (decimal 256) of **fieldflags** in **cisam_field**, and set the **colassignexpr** field in **cisam_field** to be the required expression.

An example of an expression to define a virtual column is:

```
IF FLAG<>'Y' THEN SVALUE ELSE -SVALUE
```

Where SVALUE is a real column.

Note: *When defining a virtual column which is based on other virtual columns, you must set the value of the offset column in the **cisam_field** table in the UFD file to be greater than the offsets of the virtual columns on which it is based.*

Data types

See the [C-ISAM Data Types](#) section.

Step 3: Define the Indexes

The UFD table **cisam_index** defines all the indexes associated with each **tablename** declared in the table, **cisam_table**. The following example shows the contents of the **cisam_index** UFD table:

```
cisam_index(tabname,indno,type,fld1, fld2, fld3, fld4, ..., fld64)
-----
AREA          1 U AREA_CODE
COUNTY       1 U COUNTY_CODE
CUSTOMER       1 U CUST_CODE
CUSTOMER       2 U CUST_REP      CUST_CODE
CUSTOMER       3 D CUST_REP
```

*

where:

`cisam_index` is the internal UFD table name

and the columns are:

| Column | Description |
|----------------------|---|
| <code>tabname</code> | The SQL table name that this index is for. It is the same name as in <code>cisam_table</code> . Must be UPPERCASE. |
| <code>indno</code> | The number of the index within the file. |
| <code>type</code> | U - index is unique. D - duplicate indexes allowed. |
| <code>fld1</code> | |
| ... | |
| <code>fld64</code> | Up to 64 fields are permissible in the key. You only need specify, in the template, as many index fields as are used. These key fields must be UPPERCASE. |

Note: A further column, **`indname`**, is automatically generated in the **`cisam_index`** table.

Index Sequencing

U/SQL always assumes that all index key components are in ascending sequence. However, it is possible to define C-ISAM indexes where an index key contains a component that is defined as **descending**. U/SQL does not support this and unpredictable results may occur.

To specify a descending key field, simply place a hyphen (‘-’) before it in the field name (for example, “-CUSTCODE”).

Note: If the whole of a multi-part key is descending then a hyphen is required before each part.

C-ISAM Data Types

This section lists all Universal File Dictionary (UFD) data types that can currently be used with the U/SQL C-ISAM Data Source Driver. These data types are used in a UFD text file to define the data types of individual fields, as part of an overall description of the physical layout of records within a C-ISAM data file. A UFD text file is used in the creation process of a Universal Data Dictionary (UDD).

The native C-ISAM data types are CHARTYPE, INTTYPE, LONGTYPE, FLOATTYPE, DOUBLETTYPE and DECIMALTYPE. They can be represented within a UFD text file by using the appropriate UFD data type.

The use of **high/low byte order** and **low/high byte order** refers to **big-endian** and **little-endian** respectively.

The use of **len**, **ndec** and **format** in the following tables, refers to the **cisam_field** columns in a UFD text file, where:

| | |
|---------------|--|
| len | The maximum length of the data type |
| ndec | The number of decimal places in the data type |
| format | A string comprised of a combination of key and non-key symbols, which is used to represent dates, times or timestamps. |

The following data types are supported:

- [Character Data Types](#)
- [Integer Data Types](#)
- [Floating Point Data Types](#)
- [Character Numeric Data Types](#)
- [Packed Numeric Data Types](#)
- [Miscellaneous data types](#)
- [Data Types for PowerHouse](#)
- [Date and Time Format](#)
- [Obsolescent Data Types.](#)

Character Data Types

Character data types are alphanumeric strings. The format field can be set to define date, time or timestamp.

(0 < **len** <= 254, **ndec** = 0, **format** = *format*).

| | |
|--------------|--------------------------|
| char | Non-terminated. CHARTYPE |
| charz | Null-terminated |

Integer Data Types

Integer data types are 1 to 8-byte signed/unsigned integer data types with **ndec** defining the location of an implied decimal point. The **format** field can be set to define date, time or timestamp.

($0 < \text{len} \leq 8$, $\text{ndec} \geq 0$, **format** = *format*).

| | |
|--------------|--|
| intN | Signed in native byte order |
| intL | Signed in low/high byte order |
| intH | Signed in high/low byte order INTTYPE: len = 2. LONGTYPE: len = 4 |
| uintN | Unsigned in native byte order |
| uintL | Unsigned in low/high byte order |
| uintH | Unsigned in high/low byte order |

Floating Point Data Types

Floating point data types are IEEE-754 floating point single/double precision data types.

($\text{len} = 4$ or 8 , $\text{ndec} = 0$, **format** is not applicable).

| | |
|---------------|--|
| floatN | Native byte order FLOATTYPE: len = 4. DOUBLETYPE: len = 8 |
| floatL | Low/high byte order |
| floatH | High/low byte order |

Floating point data types can also be IEEE-754 floating point single/double precision data types which are interpreted as signed integers with **ndec** defining the location of an implied decimal point.

($\text{len} = 4$ or 8 , $\text{ndec} \geq 0$, **format** is not applicable).

| | |
|--------------|---------------------|
| intfN | Native byte order |
| intfL | Low/high byte order |
| intfH | High/low byte order |

Character Numeric Data Types

Character Numeric data types are integers held as a non-terminated string with **ndec** defining the location of an implied decimal point. The **format** field can be set to define date, time or timestamp.

($0 < \text{len} \leq 15$, $\text{ndec} \geq 0$, **format** = *format*)

| | |
|------------------|----------|
| uint_char | Unsigned |
|------------------|----------|

| | |
|-------------------|-----------------------------------|
| int_charL | Leading sign (ASCII over-punch) |
| int_charT | Trailing sign (ASCII over-punch) |
| int_charLE | Leading sign (EBCDIC over-punch) |
| int_charTE | Trailing sign (EBCDIC over-punch) |
| int_charLS | Leading sign separate |
| int_charTS | Trailing sign separate. |

Character Numeric data types are numeric strings with an embedded decimal point. The decimal point is aligned as specified by **ndec**.

(0 < **len** <= 15, **ndec** >= 0, **format** is not applicable).

| | |
|-------------------|-----------------------------------|
| uflt_char | Unsigned |
| flt_charL | Leading sign (ASCII over-punch) |
| flt_charT | Trailing sign (ASCII over-punch) |
| flt_charLE | Leading sign (EBCDIC over-punch) |
| flt_charTE | Trailing sign (EBCDIC over-punch) |
| flt_charLS | Leading sign separate |
| flt_charTS | Trailing sign separate. |

Packed Numeric Data Types

4-bit binary coded decimal (BCD) (packed decimal) data types with **ndec** defining the location of an implied decimal point. Set **format** to define date, time or timestamp.

(0 < **len** <= 7, **ndec** >= 0, **format** is not applicable).

| | |
|-------------|----------|
| ubcd | Unsigned |
| bcd | Signed |

8-bit binary coded 2-digit decimal data types with **ndec** defining the location of an implied decimal point. Set **format** to define date, time or timestamp.

(0 < **len** <= 7, **ndec** >= 0, **format** is not applicable).

| | |
|---------------|----------|
| ubc2dd | Unsigned |
|---------------|----------|

Miscellaneous data types

| | |
|--------------|---|
| char1 | Non-terminated packed alphanumeric string to number conversion is base 64, with space |
|--------------|---|

| | |
|----------------|--|
| | (ASCII 32) equating to zero. |
| char3 | Non-terminated alphanumeric string with string length held in 1-byte field header. |
| Decimal | DECIMALTYPE: machine-independent up-to-17-byte floating-point (up-to-16-byte packed format mantissa, 1-bit sign and 7-bit exponent). 32 significant digits with exponents in range -128 to +126. |
| float1 | Machine-independent 6-byte floating-point (39-bit mantissa, 1-bit sign and 8-bit exponent). Low/high byte order within each 2-byte word. Exponent in excess-128 notation. Normalised mantissa with implied high-order bit of one. |
| float2 | Machine-independent 8-byte floating-point (55-bit mantissa, 1-bit sign and 8-bit exponent). High/low byte order. Exponent in excess-128 notation. Normalised mantissa with implied high-order bit of one. |
| float3 | Floating point held as a non-terminated string. Negative values are represented as either a leading or trailing minus sign. |
| fixed5E | Variation of binary code decimal (BCD) with sign held in leading nibble (0 is positive, F is negative). Number of decimal places is implied and is specified in the ndec column of the cisam_field table. |
| fixed5O | Variation of binary code decimal (BCD) with sign held in leading nibble (0 is positive, F is negative). Trailing nibble is unused. Number of decimal places is implied and is specified in the ndec column of the cisam_field table. |
| fixed8 | Integer held as a non-terminated string. Negative values are represented as an over-punch of the last digit in the range p to y, P to Y (representing values from -0 to -9). Number of decimal places is implied and is specified in the ndec column of the cisam_field table. |
| fixed9 | Machine-independent 2 to 16-byte packed signed integer with number of digits and sign held in 1-byte field header. Number of decimal places is implied and is specified in the ndec column of the cisam_field table. |
| fixed10 | Machine-independent 2 to 15-byte packed unsigned integer. Number of decimal places is implied and is specified in the ndec column of the cisam_field table. |

| | |
|----------------------------|--|
| fixed11 | Machine-independent variable length packed signed integer with sign held in trailing nibble (3 is positive, 5 is negative). Number of decimal places is implied and is specified in the ndec column of the cisam_field table. |
| fixed12 | Integer held as a fixed length null-terminated string. The string is stored right justified and space filled with the optional minus sign, '-', preceding the digits. Number of decimal places is implied and is specified in the ndec column of the cisam_field table. Fractional portion including ones digit are zero filled, that is 0 with two decimals is 000 right justified. |
| fixed13 | Fixed point held as a non-terminated string. The decimal point is contained within the string, and is positioned as specified in the ndec column of the cisam_field table. The string is stored right justified with an optional minus sign '-', and leading space characters. |
| fixed15 | Variation of binary code decimal (BCD) with sign held in trailing byte (0 to 9 for positive, 'P' to 'Y' for negative). Although ASCII over-punch characters are used to represent the sign, it does not affect the number held in BCD. Number of decimal places is implied and is specified in the ndec column of the cisam_field table. |
| money1 | Machine-dependent 2-part type comprising of a 4-byte integer, representing, say, dollars or pounds, followed by a 2-byte integer, representing cents or pence. |
| date(jd_31dec1799) | DECIMALTYPE representing Julian date since 31/Dec/1799. |
| date(t2s6_ddmmyy) | 6-byte non-terminated alphanumeric string: ddmmy1y2, with year starting at 1900. y1 is in the range 0 to 9, A to Z (representing the decades 2000 to 2250). y2 is 0 to 9. |
| date(t2s6_yymmdd) | 6-byte non-terminated alphanumeric string: y1y2mmdd, with year starting at 1900. y1 is in the range 0 to 9, A to Z (representing the decades 2000 to 2250). y2 is 0 to 9. |
| date(t2s8_dd/mm/yy) | 8-byte non-terminated alphanumeric string: dd/mm/y1y2, with year starting at 1900. y1 is in the range 0 to 9, A to Z (representing the decades 2000 to 2250). |
| date(t2s8_yy/mm/dd) | 8-byte non-terminated alphanumeric string: y1y2/mm/dd, with year starting at 1900. y1 is in the range 0 to 9, A to Z (representing the |

| | |
|-----------------------------|--|
| | decades 2000 to 2250). |
| date(t3n3f1_ymmdd) | 4-byte fixed9 representing date in the form yymmdd. |
| date(t3n4f1_ccymmdd) | 5-byte fixed9 representing date in the form ccymmdd. |
| date(t4n3f1_ymmdd) | 3-byte fixed10 representing date in the form yymmdd. |
| date(t4n4f1_ccymmdd) | 4-byte fixed10 representing date in the form ccymmdd. |
| date(t5s6_ymmdd) | 6-byte non-terminated alphanumeric string: y1 y2/mm/dd, with year starting at 1900. y1 and y2 are in the range 0 to 9, P to Y (representing 0 to 9 in the second millennium). That is, PP = 2000, PQ = 2001,... YY = 2099. |

Data Types for PowerHouse

| UFD Type | Description |
|-----------------|-----------------------------------|
| CH | CHARACTER |
| IS | INTEGER SIGNED |
| IU | INTEGER UNSIGNED |
| ZS | ZONED SIGNED |
| ZU | ZONED UNSIGNED |
| FL | FLOAT |
| PS | PACKED SIGNED |
| PU | PACKED UNSIGNED |
| FF | FREEFORM |
| JD | JDATE |
| PD | PHDATE |
| PN | PHDATE with 0 interpreted as NULL |

Date and Time Format

The CISAM_FIELD **format** column, within a UFD text file, can be used to define the format of date, time and timestamp data types. The 'base' data type can be any of the character or integer data types described previously.

A **format** string is comprised of a combination of key and non-key symbols. A Julian epoch (base) date is specified in the form "ccyymmdd". Non-key symbols, such as solidi ('/'), colons and hyphens, are considered to be stored formatting characters and are reproduced on output. If a date **format** string does not contain the "cc" century symbol, a date is considered to be within the century as determined by the date offset ini file entry, **FixedDateOffset**. The key symbols are described in the table below:

| Key Symbol | Description |
|-------------------|---|
| CC | 2-digit century. |
| YY | 2-digit year. |
| MM | 2-digit month. |
| MON | 3-character month. (i.e. Jan, Feb, Mar, ...) |
| DDD | 3-digit days since start of year. |
| DD | 2-digit day. |
| J | Julian date (in days). Epoch date follows symbol. |
| hh | 2-digit hour (24-hour format). |
| mmmm | 4-digit minutes since midnight. |
| mm | 2-digit minute. |
| sssss | 5-digit seconds since midnight. |
| ss | 2-digit second. |
| ff | 2-digit fraction of second. |
| jh | Julian timestamp in hours. Epoch date follows symbol. |
| jm | Julian timestamp in minutes. Epoch date follows symbol. |
| js | Julian timestamp in seconds. Epoch date follows symbol. |

Examples of date format strings are:

"CCYYMMDD", "YYDDD", "J19601202", "DD/MON/CCYY".

Examples of time format strings are:

"hhmmss", "sssss", "hh:mm:ss".

Example of timestamp format strings are:

"jm19601202", "CCYY/MM/DD-hh:mm:ss:ff".

Obsolescent Data Types

The table below lists the obsolescent data types and the new data types along with the requisite format strings (where applicable), they have been replaced by:

| Obsolescent data type name | New data type name | Length | Format string |
|-----------------------------------|---------------------------|---------------|----------------------|
| char | char | - | Not applicable |
| char2 | charz | - | Not applicable |
| date(i4_yyyyymmdd) | intN | 4 | CCYMMDD |
| date(j3H_31dec1599) | intH | 3 | J15991231 |
| date(j4_31dec1899) | intN | 4 | J18991231 |
| date(j4H_31dec1899) | intH | 4 | J18991231 |
| date(j4L_31dec1899) | intL | 4 | J18991231 |
| date(js_31dec1899) | char | - | J18991231 |
| date(js6_31dec1899) | char | 6 | J18991231 |
| date(n10_mm-dd-ccyy) | char | 10 | MM-DD-CCYY |
| date(s6_ddmmyy) | char | 6 | DDMMYY |
| date(s6_mmddy) | char | 6 | MMDDYY |
| date(s6_yymmdd) | char | 6 | YYMMDD |
| date(s8_ccyymmdd) | char | 8 | CCYMMDD |
| date(s8_dd/mm/yy) | char | 8 | DD/MM/YY |
| date(s8_ddmmccyy) | char | 8 | DDMMYYCC |
| double | floatN | 8 | Not applicable |
| doubleH | floatH | 8 | Not applicable |
| doubleL | floatL | 8 | Not applicable |
| fixed14 | intN | 2 | Not applicable |
| fixed14H | intH | 2 | Not applicable |
| fixed14L | intL | 2 | Not applicable |
| fixed3 | intfN | 8 | Not applicable |
| fixed3H | intfH | 8 | Not applicable |

| | | | |
|-----------------------|------------|---|----------------|
| fixed3L | intfL | 8 | Not applicable |
| fixed4 | intN | 4 | Not applicable |
| fixed4H | intH | 4 | Not applicable |
| fixed4L | intL | 4 | Not applicable |
| fixed6 | flt_charLS | - | Not applicable |
| fixed7 | int_charT | - | Not applicable |
| float | floatN | 4 | Not applicable |
| floatH | floatH | 4 | Not applicable |
| floatL | floatL | 4 | Not applicable |
| int/int2H | intH | 2 | Not applicable |
| int1 | intN | 1 | Not applicable |
| int2 | intN | 2 | Not applicable |
| int2L | intL | 2 | Not applicable |
| int4 | intN | 4 | Not applicable |
| int4L | intL | 4 | Not applicable |
| int5L | intL | 5 | Not applicable |
| long/int4H | intH | 4 | Not applicable |
| numchar | char | 2 | Not applicable |
| time(m2H) | intH | 2 | mmmm |
| time(m2L) | intL | 2 | mmmm |
| tstamp(m4H_31dec1899) | intH | 4 | jm18991231 |
| tstamp(m4L_31dec1899) | intL | 4 | Jm18991231 |
| uint1 | uintN | 1 | Not applicable |
| uint2 | uintN | 2 | Not applicable |
| uint2H | uintH | 2 | Not applicable |
| uint2L | uintL | 2 | Not applicable |
| uint3L | uintL | 3 | Not applicable |
| uint4 | uintN | 4 | Not applicable |
| uint4H | uintH | 4 | Not applicable |

| | | | |
|--------|-------|---|----------------|
| uint4L | uintL | 4 | Not applicable |
|--------|-------|---|----------------|

Support for Record Number Access

When a field with a type (field type) of 'recno' is added to the **cisam_field** table, an internal U/SQL pseudo index is automatically provided to allow data access by record number. The offset of this field positions it with relation to other fields. It does not have to be mapped over blank data. You cannot change the value of the record number field, and it is ignored when inserting or updating. You must only define one of these fields per table.

For example, consider the following UFD text file:

```

cisam_table  (tabname,  reclen,  nfield,  nindex,  data_file,  recexpr)
              -----  -----  -----  -----  -----  -----
                    TEST1      80      4      0      TEST1      NULL

*

cisam_field  (tabname,  fldname,  type,  length,  offset,  ndec)
              -----  -----  ----  -----  -----  -----
                    TEST1      RECID   recno  0      0      0
                    TEST1      FLD1_1  int   2      0      0
                    TEST1      FLD1_2  int   2      2      0
                    TEST1      FLD1_3  int   2      4      0
                    TEST1      TEXT1   char  14     6      0

*

cisam_index (tabname, indno, type, fld1, fld2, fld3, fld4)

```

Note:

- An entry for **RECID** is not required in the 'index' table as it is a pseudo index
- The **RECID** field is needed in **INSERT** and **UPDATE** statements, and the data entry in the table must be set to **NULL**
- In the example above the file contains no C-ISAM indexes. If it did then entries are needed in the **cisam_index** table for the 'real' index fields.

Typical queries are:

```
SELECT * FROM test1 WHERE RECID > 8;
```

or:

```
INSERT INTO test1 VALUES ("",1,1,1,"Insert");
```

```
SELECT * FROM test1 WHERE FLD1_3 BETWEEN 0 and 3;
```

Setting Up a Business BASIC Data Dictionary

Setting Up a Business BASIC Data Dictionary

This section describes setting up a UDD for Business BASIC (BBASIC) data. Universal Business Language (U/BL) and its ISAM, derived from Universal Business BASIC (UBB), is Transoft's Open Systems version of Business BASIC. It is compatible with Data General's Business BASIC, B32 Business BASIC and Bluebird's SuperDOS Business BASIC.

This section applies only to Multiple-tier installations for both UNIX and Windows NT Server platforms.

Windows NT Server

Note that Windows NT Server Revision 3.00 of U/SQL for BBASIC has the following restriction:

- There is no support for BBREUSE and Extended File System (EFS).

Note: Refer to the [Overview](#) and [Creating a UDD](#) sections for an overview of the dictionary technology and the steps involved in creating a UDD.

Introduction

The U/SQL Server supports a data source driver for BBASIC ISAM covering all the main file types, for example, Logical, PARAM-based or Linked Lists via R1 pointers. In addition, all the main data types are supported.

The first step is to [define the contents of your data dictionary](#).

Defining a BBASIC ISAM Data Dictionary

Create a Text File

The data dictionary is specified by creating a text file, for example, **dictionaryname.ufd**. It consists of four sections, each of which relates to a table in the UFD:

- **bb_logfile**
- **bb_table**
- **bb_field**
- **bb_index**

Each table is specified as an entity, and a table type can be repeated as any number of entities.

How you specify this UFD text file is described in the [Creating a UDD](#) section.

The following steps must be performed to create tables to specify your U/BL and UBB data files and their structure:

- [Step 1: Define the Data Files](#)
- [Step 2: Point the SQL Table Names to the BBASIC Logical Filenames](#)
- [Step 3: Define the Fields \(Columns\)](#)
- [Step 4: Define the Indexes.](#)

Step 1: Define the Data Files

```
bb_logfile (lfname,ftype,dbfile,start_sector,reclen,last_record)
-----
AREAS      L  AREADATA          0   24    50
COUNTIES L  COUNTY            0   22   100
CUST       L  CUSTMAST          8  204  5000
TRANS      L  TRANSMAST         0  200 10000
*
```

where:

`bb_logfile` is the internal UFD table name

and the columns are:

| Column | Description |
|--------------|---|
| lfname | The logical or sub-file name. This is the name used in a BBASIC program in either a D\$ string or LOPEN statement. |
| ftype | The file type, see below. |
| dbfile | The physical file name. |
| start_sector | Starting sector of the logical file within the physical file. |
| reclen | The record length. |

| | |
|-------------|--|
| last_record | The number of records in the logical file. |
|-------------|--|

File types, *ftype*, are:

| ftype | Description |
|--------------|---|
| D | Direct Access, for example, a control file, where record zero contains data. |
| L | Linked available record. A normal BBASIC ISAM file on which you can perform GETREC, and so on, where there is a record zero that is not a data record. |
| PD, PL | It is also possible to obtain the equivalent of the D and L file type information directly from the PARAM file by setting ftype to PD or PL. In this case start_sector , reclen and last_record must be set to zero. |
| VD, VL | These are equivalent to the D and L types above, except that they specify use of BBASIC logical file volume label information directly. In this case start_sector , reclen and last_record must be set to zero. |
| SD | SuperDOS FCB file. In this case start_sector must be set to '2'. Set reclen and last_record as usual, as these entries are not read from the FCB. |
| N | THIS MUST BE USED READ ONLY. This is a special case and means that the R1 pointer will be used to calculate the record position by applying the algorithm of: $\text{Record_Posit}'n = \text{Start_sector} + 512 + ((R1 - 1) * \text{Record_length})$. |
| X | THIS MUST BE USED READ ONLY. This is a special case and means that the R1 pointer will be used as an absolute offset, rather than a record number. |
| Z | THIS MUST BE USED READ ONLY. Deleted records are flagged with a status of 0. The file type Z means that the file is a "linked available record" type file, that is normally a type L, but that a record in the file is only considered to be logically deleted if its first two bytes are equal to zero. Records with non-zero values in their first two bytes are considered to be active. This follows the B32 Business Basic convention for "linked available record" type files. In DG Business BASIC, UBB and U/BL, a record is considered to be logically deleted if its first two bytes are less than or equal to zero. |
| R | Should only be set for an Index file. This has the same effect for the specific file as setting the BBREUSE INI directive, for all index files . It allows empty index blocks to be reused by the maintenance of an empty block chain in record zero of the files. If BBREUSE is set to Y, in the U/SQL INI file, the BB-ISAM file handler will manage the empty index block chain |

| | |
|---|---|
| S | If this is set for an Index file, the index key will be space filled rather than null-filled. |
| A | If this is set for a data file, all records will be read when accessing the data file with a sequential read, without checking for deleted records. |

Step 2: Point the SQL Table Names to the BBASIC Logical Filenames

It is often convenient for the SQL table names to be the same as the BBASIC logical filenames.

```
bb_table (tablename, lfname, tabtype, cond)
```

```
-----
AREAS          AREAS
COUNTIES     COUNTIES
CUSTOMERS     CUST
INVOICES      TRANS          REC_TYPE="I"
INDEX         TYPEIX          I
*
```

where:

bb_table is the internal UDD table name

and the columns are:

| Column | Description |
|-----------|--|
| tablename | The SQL table name, that is, the name the ODBC-enabled products will 'see'. Table names must be SQL compliant and satisfy the following conditions: <ul style="list-style-type: none"> • are in UPPERCASE • are within 18 characters • are unique • are not SQL keywords • are not zero length • are made up of 0-9, A-Z, _, with the first character alphabetic and with no blanks <p>Note: To ensure consistent locking with your U/BL applications, tabnames need to have the first 14 characters unique and be the same names as used in U/BL lock statements.</p> |
| lfname | The BBASIC logical or sub-file name. This is the name that matches the lfname field in the bb_logfile . |
| tabtype | The table type: see below. |
| cond | Conditional test to establish if record belongs to table. This is an expression (maximum 250 characters). See the Multiple Record Type Handling section below. |

Index Names Note: When logical filenames used for indices have "." in their names, care must be taken to ensure that the length of the filename up to the "." is the same for all indices, or at least not relied upon for order. This is because U/SQL translates the "." to an underscore ("_") since a "." in an index-name is invalid. However, an underscore is above "Z" in ASCII order, so two indices like "MYINDEX.IX" and "MYINDEX2.IX" would be in correct order before substitution, but after substitution "MYINDEX_IX" would actually be after "MYINDEX2_IX". This causes an error as the wrong index can be used for a query and the wrong results generated.

Valid table types, `tabtype`, are:

| tabtypes | Description |
|-----------------|--------------------|
| I | Index file |
| D or blank | Data File |

Multiple Record Type Handling

The **cond** column in the **bb_table** can contain a condition or expression. Only records that satisfy the condition or expression will be included in the table. The syntax of the conditional expression is exactly the same as that for an SQL **WHERE** clause **search-condition** (see the [Expression Handling](#) section) and refers to columns in the table.

For example, assume the file **TRANSMAST** contains two record types, INVOICES and PAYMENTS. They are distinguished by INVOICES having the field (column) TYPE=1 and PAYMENTS having TYPE=2 AND CREDIT="Y". You could therefore define three tables called, say, TRANSACTION, INVOICE and PAYMENT, which will be 'seen' to contain all transactions, just invoices and just payments respectively, when viewed by an ODBC-enabled product. The entries in the **bb_table** are:

```
bb_table(tabname, lfname, tabtype, cond)
-----
TRANSACTION TRANSMAST      D
INVOICE      TRANSMAST      D TYPE=1
PAYMENT      TRANSMAST      D TYPE=2 AND CREDIT="Y"
*
```

Step 3: Define the Fields (Columns)

```
bb_field (tabname, fldname, type, len, offset, ndec)
-----
AREAS      ARCOD      A      4      1      0
AREAS      ARDES      A     20      5      0
COUNTIES COCOD      I      2      1      0
COUNTIES CDESC      A     20      3      0
CUSTOMERS SPARE      A      2      1      0
CUSTOMERS CCODE      A      8      3      0
CUSTOMERS CADD      C     80     11      0
CUSTOMERS AREA      I      1     96      0
CUSTOMERS REPCODE   L      1     97      0
CUSTOMERS CCAT      I      1     98      0
CUSTOMERS PNTFTR    L      4     99      0
```

```
CUSTOMERS  DTELST      J      2   103   0
CUSTOMERS  RECNO      R           150
*
```

where

bb_field is the internal UFD table name

and the columns are:

| Column | Description |
|-----------|---|
| tablename | The SQL table name, that is, the name that the ODBC-enabled products will 'see'. Table names must be SQL compliant and satisfy the following conditions: <ul style="list-style-type: none"> • are in UPPERCASE • are within 18 characters • are unique • are not SQL keywords • are not zero length • are made up of 0-9, A-Z, __, with the first character alphabetic and with no blanks |
| fldname | The field (column) name. Column names must be SQL compliant and satisfy the following conditions: <ul style="list-style-type: none"> • are in UPPERCASE • are within 18 characters • are unique • are not SQL keywords • are not zero length • are made up of 0-9, A-Z, __, with the first character alphabetic and with no blanks |
| type | The data type. |
| len | The data length. |
| offset | The offset within the record. |
| ndec | The number of decimal places. |

Valid data types, type, are:

| type | Description |
|------|---|
| 3 | THIS MUST BE USED READ ONLY. Pseudo triple precision variable, that is the first two bytes hold an amount which will be multiplied by 100,000,000 and the next four bytes hold an amount which is added to the first result. Set Reclen to 6 and it is possible to have a ndec value. |

| | |
|---|--|
| A | Alphanumeric (null filled). |
| S | <p>The data type S should be used for alphanumeric fields which are involved in multi-component keys, and which have been padded with space characters in the Business Basic application.</p> <p>Normally, the U/SQL BBASIC ISAM Server expects such fields to be padded with nulls. Defining the fields as data type A causes records to be mistakenly flagged as deleted when viewed using third party ODBC-enabled products such as Microsoft Access. Therefore, define the fields as data type S so that the U/SQL Server is aware that they have been padded with spaces rather than nulls.</p> <p>Note: <i>The S data type only applies to alphanumeric fields which are space filled and involved in multi-component keys.</i></p> |
| B | Bit (For this type, ndec specifies the relevant bit 0-n). |
| C | Crammed. |
| I | Integer (signed). |
| J | Julian date; normal DG format. A zero value gives a date of "0000-01-01" which is considered a "valid" date. For example, Microsoft Access will accept this date. |
| K | Julian date. A zero value gives a date of "0000-00-00" which is NOT considered a "valid" date. For example, Microsoft Access will NOT accept this date. |
| D | <p>Numeric date of format YYMMDD or YYYYMMDD determined by the length.</p> <ul style="list-style-type: none"> • For 8 digit dates, set ndec to 0. • For 6 digit dates, ndec is overloaded to specify the two digit 'switchover' year for the implied century, where: <ul style="list-style-type: none"> - 0=YY<=ndec implies year = 20YY - ndec<YY implies year = 19YY <p>For example, if ndec=25 then 050717='2005-07-17' and 320717='1932-07-17'.</p> |
| E | <p>Numeric date of format DDMMYY or DDMMYYYY determined by the length.</p> <ul style="list-style-type: none"> • For 8 digit dates, set ndec to 0. • For 6 digit dates, ndec is overloaded to specify the two digit 'switchover' year for the implied century, where: <ul style="list-style-type: none"> - 0=YY<=ndec implies year = 20YY - ndec<YY implies year = 19YY <p>For example, if ndec=25 then 170705='17-07-2005' and 170732='17-07-1932'.</p> |
| L | Linked list pointer. This denotes that the entry in this field is the R1 pointer to a record in the same or another file. Use only as |

| | |
|---|---|
| | join in WHERE clause. (0 is the end of list). |
| R | Record number of the current row (only one column per table can be this type). See the Record Number Column section. |
| U | Unsigned integer. |
| F | Numeric date of format MMDDYY. This displays a similar behaviour to the existing date data types D, and E. You can have either a 8-digit date when ndec =0 or a 6 digits date when ndec > 0. |
| H | Numeric date of format MMDDYYY. This has a 3-digit number for the year. This means the year is actually the addition 1900+YYY. For example: 0105032 ==>05/01/1932 (dd/mm/yyyy) 0105132 ==>05/01/2032 (dd/mm/yyyy) |
| M | Numeric date of format DDMMYYY. This has a 3-digit number for the year. This means the year is actually the addition 1900+YYY. For example: 0501032 ==>05/01/1932 (dd/mm/yyyy) 0501132 ==>05/01/2032 (dd/mm/yyyy) |
| N | Numeric date of format YYMMDD. This has a 3-digit number for the year. This means the year is actually the addition 1900+YYY. For example: 0320105 ==>05/01/1932 (dd/mm/yyyy) 1320105 ==>05/01/2032 (dd/mm/yyyy) |

Note: SuperDOS special cram characters are supported.

Step 4: Define the Indexes

Defining the indexes is a two-stage process. First, you define the index filename in the **bb_logfile** table. This is similar to the data file definition in [Step 1](#):

```
bb_logfile (lfname,ftype,dbfile,start_sector,reclen,last_record)
-----
CUSTINDEX I CUSTMAST 0 512 400
*
```

where:

bb_logfile is the internal UDD table

and the columns are:

| Column | Description |
|--------|---|
| lfname | The logical or sub-file name. This is the name used in a BBASIC program in either a D\$ string or LOPEN statement. |

| | |
|--------------|---|
| ftype | The file type, see below. |
| dbfile | The physical file name. |
| start_sector | The starting sector of the logical file within the physical file. (Each sector is always 512 bytes regardless of index block size). |
| reclen | Record length of the index block (512 or 2048). |
| last_record | The number of records in the logical file. |

File types, *ftype*, are:

| ftype | Description |
|------------|--|
| I or blank | Normal index. |
| PI | To obtain the equivalent index file type information directly from the PARAM file, set ftype to PI . In this case, set start_sector , reclen and last_record to zero. |
| VI | To obtain the equivalent index details from the BBASIC logical file volume label directly, set ftype to VI . In this case, set start_sector , reclen and last_record to zero. |

Secondly, point the SQL table at the index file and state the key fields:

```
bb_index (tablename, indname, indtype, fld1, fld2, fld3, fld4, . . . . ., fld64)
```

```
-----
```

```
CUSTOMERS  CUSTINDEX  U  CCODE
```

```
*
```

where:

bb_index is the internal UFD table name

and the columns are:

| Column | Description |
|-----------|---|
| tablename | The SQL table name; that is, the name that the ODBC-enabled products will 'see'. Table names must be SQL compliant, see bb_table. They must be UPPERCASE. |
| indname | The index name as defined above. |
| indtype | Index type, see below. |
| fld1 | The first field in the key. |
| ... | |
| fld64 | Up to 64 fields are permissible in the key. You only need specify, in the template, as many index fields as are used. These key |

| | |
|--|---------------------------|
| | fields must be UPPERCASE. |
|--|---------------------------|

Index types, `indtype`, are:

| indtype | Description |
|----------------|---------------------|
| D | Duplicates allowed. |
| U | Unique. |

Record Number Column

A record number column can be added to all tables. If the column appears in a table defined on a data record, it will contain the record number of that data record. When the record number column is contained in a table defined on an index, it will contain the record number (R1) associated with the key in the index.

If the record number column appears in a table defined on a data record, an index using the record number will automatically be defined in the UDD.

The record number column can be added to a table by adding an extra field in the **bb_field** table. This field must have its **type** set to **R**. The **len** and **ndec** fields do not have to be set. Since columns are arranged in a table in field offset order, the **offset** column can be used to position the record number column within the table. For example, set the offset to 0 if the column should be first in the table, or set it to something greater than the last data field offset, to make it appear last in the table.

Tables Defined only on Data Records

Normally you will define tables for data records with their associated indexes, that is [Step 2](#) and [Step 4](#) above, so that as a data record is added or deleted its index entries are automatically updated accordingly.

However, you can simply define a table for data records and not include the associated index(es), that is do not include [Step 4](#) (**bb_index**). If a record from such a table is deleted, only the data record is deleted: any index must be deleted separately. If a new record is inserted, only the data record is inserted and no index.

Note: *If the table contains a record number column, see the [Record Number Column](#) section above, an index on it will be automatically created. This means the table still has a unique key.*

Tables Defined Only on Indexes

Tables can be defined on an index to provide access, key insertion and deletion independent of any associated data file. To achieve this, set the field in **bb_table** called **tabtype** to the letter **I**. Set the logical filename field, **lfname**, to the filename that contains the index. The index for any such table, defined in the **bb_index** table, will of course contain all the fields in the table (except any record number field), see the [Record Number Column](#) section. For example:

```
bb_table (tablename, lfname, tabtype, cond
-----
INDEX      TYPEIX      I
```

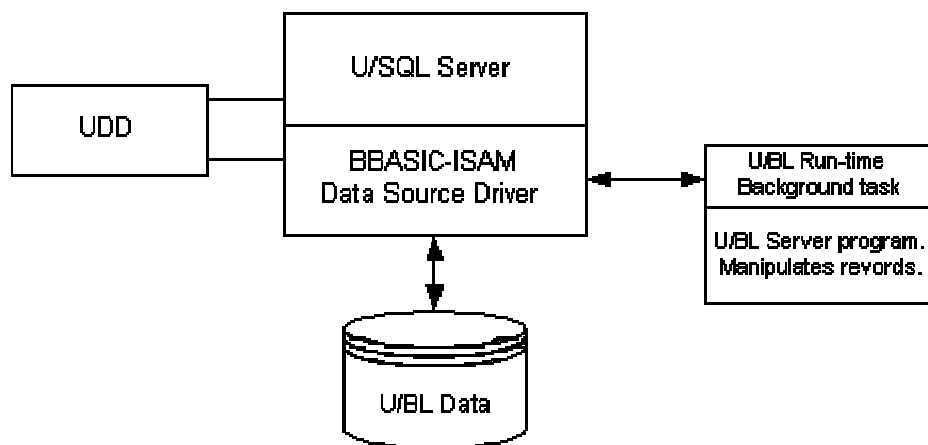
Note: *Do not define a table on an index unless you specifically require this independent access to the index. Indexes set up in this way are not automatically updated with the associated data record.*

U/SQL Record Mapper

Business BASIC allows you to create your own data types that are not directly supported by U/SQL. These extensions also cover, for example, special cram characters, numeric values where the decimal point is implied by another 'controlling' field, and so on.

This functionality is facilitated by the U/SQL Record Mapper. The U/SQL Record Mapper allows the manipulation of BBASIC ISAM data by a UBB or U/BL program prior to its submission to the U/SQL Server engine. This means that data types which are not currently supported by U/SQL can be manipulated into supported data types. Additionally it allows for "virtual columns", that is data items to be calculated or translated from other fields in the BBASIC ISAM file.

The diagram below shows the U/SQL Record Mapper architecture:



The following example makes use of the [Demonstration Books Wholesaler](#) application, provided with U/SQL, to illustrate these features.

Note: This example assumes a prior knowledge of UBB and U/BL programming and the maintenance and creation of U/SQL UDDs.

There are three distinct phases in setting up the U/SQL Record Mapper:

- [Define the remapped table](#) in the UDD via modification to the UFD. The UFD, **remap.ufd**, is provided as an example.
- [Provide the U/SQL Server with the location of the message queue.](#)
- [Create a UBB or U/BL program](#), which can be run as a background job, to handle the remapping. This communicates with the U/SQL Server via a message queue. The UBB or U/BL program, **remap.bb**, is provided as an example.

Defining the Remapped Table

Within **remap.ufd** there are three remapped tables:

| Original table name | Remapped table name | Feature illustrated |
|---------------------|---------------------|---------------------|
| | | |

| | | |
|----------|----------|------------------------|
| ORDERS | NEWORDER | Data type manipulation |
| STOCK | NEWSTOCK | Virtual Columns |
| CUSTOMER | NEWCUST | Data translation |

Firstly, we shall look in detail at the ORDERS and NEWORDER definition.

There are no additional changes to **bb_logfile** but in **bb_table** there is an additional table NEWORDER, which is linked to the **lfname**, ORDERS.

```
bb_logfile (lfname,dbfile,start_sector,reclen,last_record)
```

```
-----
ORDERS      BOOKS          291      32          100
ORDRIX      BOOKS          298     512           50
*
```

```
bb_table (tabname,lfname)
```

```
-----
ORDERS      ORDERS
NEWORDER    ORDERS
*
```

Within **bb_field** we now have entries for both tables. The item we wish to remap is ODATE. In the underlying data ODATE is held as a 4 byte numeric in the format YYMMDD, for example 970728 (28th July 97). We wish to remap this to a DG Julian date. All other fields remain the same but ODATE is type I in table ORDERS whereas it is type J in table NEWORDER. Note that the record length of ORDERS is 32 but NEWORDER.ODATE is defined with offset 33.

Note: *It is the definition of an offset greater than the record length that will trigger remapping.*

Additional remapped fields can be added to NEWORDER, if required. Since the length of NEWORDER.ODATE is defined as length 4 then the next remapped field would have offset 37, and so on for any extra fields that you may add.

```
bb_field (tabname, fldname, type, len, offset, ndec)
```

```
-----
ORDERS      ORDNO          I         4         3         0
ORDERS      CUSTCODE       A         4         7         0
ORDERS      ODATE          I         4        11         0
NEWORDER    ORDNO          I         4         3         0
NEWORDER    CUSTCODE       A         4         7         0
NEWORDER    ODATE          J         4        33         0
*
```

The final amendment is to duplicate the index definition(s) of table ORDERS for table NEWORDER.

```
bb_index (tabname, indname, indtype, fld1, fld2)
```

```
-----
ORDERS      ORDRIX         U         ORDNO
NEWORDER    ORDRIX         U         ORDNO
*
```

The UDD can now be created, from this UFD as described in the Creating the UDD from the UFD Text File section.

Provide U/SQL Server with the Location of the Message Queue

The location of the message queue is determined via the **UBLMAPSVR** directive in the UNIX **usqlsd.ini** configuration file or the Windows NT Server Registry. It can be a global setting in the **[Data Source Defaults]** section or local to a particular UDD. Example UNIX **usqlsd.ini** entries are:

```
[Data Source Defaults]
UBLMAPSVR=/u2/proj/client_data/bridge/UBLMAPPER
[remap.udd]
UBLMAPSVR=/tmp/UBLMAPFILE
```

The **UBLMAPSVR** file provides the location of the message queue, in memory. It is not the conduit for the messages.

Note: Configuration directives are described in the [OBDC.INI Directives](#) section.

Create U/BL Program to Service the Remapping

The sample program **remap.bb**, shown below, is a guide only. Since it contains additional information, which is displayed to the screen for illustrative purposes, it must be run in the foreground on a terminal session. In a live situation you will normally run the program as a background job. The basic purpose of the program is to create, and manage, a message queue between itself and the U/SQL Server process.

We shall now look at its component parts in detail, with particular reference to the tables **ORDERS**, and **NEWORDER**. You may find it useful to refer to the *U/BL Online Help* for more background on the **STME** calls which are made.

```
00001 REM Example U/SQL Record Mapping Server
00010 DIM SERVER$(80)
00020 LET SERVER$="/usr/client_data/bridge/UBLMAPPER"
```

Note that **SERVER\$** corresponds to the **UBLMAPSVR** directive entry.

```
00100 GOSUB 10000 : initialize tables
00110 GOSUB 20000 : Create the message Q. Expects a variable SERVER$
00120 PRINT "Waiting for messages on global port ";GPORT
00130 STMA 8,5
10000 REM initialize tables we can deal with
```

Obviously there are many ways in which this can be done.

```
10010 LET MAXTAB=10
10020 DIM TABNAM$(MAXTAB*30),T$(30),MODE$(1)
10030 RESTORE 10200
10040 LET NTAB=0 \ TABNAM$=""
10050 READ T$
10060 IF T$="*" THEN GOTO 10110
10070 LET T$(0)=FILL$(0)
10080 LET TABNAM$(0)=T$
10090 LET NTAB=NTAB+1
10100 GOTO 10050
10110 RETURN
10200 DATA "NEWCUST"
10210 DATA "NEWSTOCK"
10220 DATA "NEWORDER"
10290 DATA "*"

20000 REM Create the message Q. Expects a variable SERVER$
20010 DIM MSGCTRL$(24),MSGBUF$(4096),LPUSED(100)
20020 LET LPORT=1 \ GPORT=0
```

Transoft U/SQL User Guide

```
20030 STME 23,E,SERVER$,LPORT
20040 IF E<>-1 THEN GOTO 29000 : error condition
```

A local port for communications has now been established.

```
20050 STME 26,E,LPORT,GPORT
20060 IF E<>-1 THEN GOTO 29000 : error condition
```

A global port is now bound the local port and GPORT will be used to send messages to, and receive them from, the U/SQL Server.

```
20070 LET MSGCTRL$=FILL$(0)
20080 LET MSGBUF$=FILL$(0)
20090 FOR I=1 TO 100
20100   LET LPUSED[I]=0
20110 NEXT I
```

LPUSED is an array which keeps track of the user number.

```
20990 RETURN
```

```
00200 GOSUB 21000 : receive a message
00202 PRINT "rcvd: ";MSGBUF$[1,31]
00203 PRINT
00204 STMA 8,5
```

```
21000 REM receive a message
21010 LET MSGCTRL$=CHR$(0,4),CHR$(1,4),CHR$(LPORT,2),FILL$(0)
21020 STME 21,E,MSGCTRL$,MSGBUF$
21030 IF E<>-1 THEN GOTO 29000 : error condition
21040 LET CLNTPID=ASC(MSGCTRL$[3,4])
```

CLNTPID will be one of the following values:

- CLNTPID=1 - A new connection is required by an attaching U/SQL Server process, that is, a new client is connecting.
- CLNTPID=2 - A client has disconnected.
- CLNTPID>2 - This signifies that the message relates to a remapping.

```
21050 IF CLNTPID=1 THEN
21060   GOSUB 21500 : allocate a free local port for new client
21070   GOTO 21010
21080 END IF
21090 IF CLNTPID=2 THEN
21110   GOSUB 21700 : release a local port (client done)
21120   GOTO 21010
21130 END IF
```

Otherwise return to process the message.

```
21190 RETURN
```

```
21500 REM allocate a free local port for new client
21510 FOR I=1 TO 100
21520   IF LPUSED[I]=0 THEN GOTO 21540
21530 NEXT I
21540 LET LPUSED[I]=1
21550 LET MSGBUF$=CHR$(I+2,2)
```

Because values 1 and 2 are used for other purposes (see above), I+2 is the CLNTPID.

```
21560 GOSUB 22000 : send a message
21562 PRINT "alloc client ";I+2
21563 STMA 8,5
```

```
21570 RETURN

21700 REM release a local port (client done)
21710 LET I=ASC(MSGBUF$[1,2])
21720 LET LPUSED[I-2]=0
```

Again we must allow for CLNTPID values of 1 and 2.

```
21722 PRINT "release client ";I
21724 STMA 8,5
21730 RETURN

22000 REM send a message
22020 LET MSGCTRL$=CHR$(0,4),CHR$(GPORT,4),CHR$(CLNTPID,2),FILL$(0)
22030 STME 20,E,MSGCTRL$,MSGBUF$
22040 IF E<>-1 THEN GOTO 29000 : error condition
22050 RETURN
```

Now look at how an input message is processed.

```
00200 GOSUB 21000 : receive a message
00202 PRINT "rcvd: ";MSGBUF$[1,31]
```

The input message buffer contains:

| | | |
|------------|---|---|
| byte 1 | mode | I = input mapping (from a SELECT) O = output mapping (UPDATE, or INSERT) |
| bytes 2-31 | tablename | |
| bytes 32 | to end of buffer contains the record contents | |

```
00203 PRINT
00204 STMA 8,5
00210 GOSUB 11000 : identify table
```

The table named is checked to be known to the program, before the code is executed that processes it to your requirements.

```
00220 ON TABNO THEN GOSUB 01000,02000,03000
```

Firstly, we shall examine a **SELECT** on the NEWORDER table (TABNO=3) and, after the remapping, send a message back to the U/SQL Server.

```
00230 GOSUB 22000 : send a message
00240 GOTO 00200 To get the next message
```

```
11000 REM identify table
11010 LET MODE$=MSGBUF$[1,1] MODE$ can be "I" or "O" (See above)
11020 LET T$=MSGBUF$[2,31]
11030 FOR TABNO=1 TO NTAB
11040 LET N=(TABNO-1)*30+1
11050 IF TABNAM$[N,N+29]=T$ THEN RETURN
11060 NEXT TABNO
11070 PRINT "Table ";T$;" is not known"
11080 STOP
```

```
03000 REM Mapping for NEWORDER table
03010 DIM ORD$[32],NEWORD$[36],R$[6],RN$[20]
03020 IF MODE$="O" THEN GOTO 03500 : output mapping of NEWORDER
```

In this instance MODE\$ will be "I" (To satisfy a SELECT statement). Therefore we will manipulate ODATE from YYMMDD to a DG Julian date.

Transoft U/SQL User Guide

```
03030 LET ORD$=MSGBUF$[32,61]
```

Initially we read in the UNMAPPED record and, since the record length of the ORDER table is 64, we extract bytes 32 to 95 of the message buffer. Refer to the previously described layout of the buffer.

```
03040 LET DATE=ASC(ORD$[11,14])
```

If you refer back to the original definition of ORDERS.ODATE in **remap.ufd** you will see that it is a four byte integer at offset 11 and this is what is extracted into the variable DATE.

```
bb_field (tablename, fldname, type, len, offset, ndec)
```

```
-----
```

| tablename | fldname | type | len | offset | ndec |
|-----------|---------|------|-----|--------|------|
| ORDERS | ODATE | I | 4 | 11 | 0 |

```
03050 LET YY=DATE/10000
03060 LET MM=MOD(DATE/100,100)
03070 LET DD=MOD(DATE,100)
03080 STMA 12, JDATE, MM, DD, YY
```

The year, month, and day components of DATE are extracted and an **STMA 12** used to place the Julian date in JDATE.

```
03090 LET JDATE=CHR$(JDATE,4)
03110 LET NEWORD$=ORD$,RN$
```

JDATE is then placed in a return string JDATE and the remapped record NEWORD\$ is created from the original ORDER record in ORD\$ (length 32 bytes) plus RN\$. Thus the offset of NEWORDER.ODATE in table NEWORDER is 33 with a length of 4 bytes. This is reflected in **remap.ufd** as:

```
bb_field (tablename, fldname, type, len, offset, ndec)
```

```
-----
```

| tablename | fldname | type | len | offset | ndec |
|-----------|---------|------|-----|--------|------|
| NEWORDER | ODATE | J | 4 | 33 | 0 |

and will be unpacked from the returned message buffer by the U/SQL Server.

```
03120 LET MSGBUF$[32]=NEWORD$
03130 RETURN
```

The method of sending the message is as follows:

```
22000 REM send a message
22020 LET MSGCTRL$=CHR$(0,4),CHR$(GPORT,4),CHR$(CLNTPID,2),FILL$(0)
```

GPORT was obtained from:

```
20050 STME 26,E,LPORT,GPORT
```

CLNTPID is extracted from the input message

```
21040 LET CLNTPID=ASC(MSGCTRL$[3,4])
```

and the message is sent with a **STME 20**

```
22030 STME 20,E,MSGCTRL$,MSGBUF$
22040 IF E<>-1 THEN GOTO 29000 : error condition
22050 RETURN
```

The net result of this can be seen by comparing the results of a SELECT on the tables ORDERS, and NEWORDER. For example, using the UNIX Interactive U/SQL utility, **usqli**:

```
U/SQL> select * from orders;
  ORDNO  CUSTCODE      ODATE
  -----
  123001 C001              940723

U/SQL> select * from neworder;
  ORDNO  CUSTCODE      ODATE
```



```

-----
123001 C001          1994-07-23
-----

```

Whilst ORDERS.ODATE is simply a number, NEWORDER.ODATE is correctly treated as a date.

We will now examine the remapping which applies in an UPDATE, or INSERT on the NEWORDER table. We are assuming the receipt of a message with mode "O".

```

03020 IF MODE$="O" THEN GOTO 03500 : output mapping of NEWORDER
03500 REM output mapping of NEWORDER
03510 LET NEWORD$=MSGBUF$(32,67)

```

We firstly extract the 36 bytes of the record NEWORDER into a string NEWORD\$.

```

03520 LET ORD$=NEWORD$(1,32)

```

The first 32 bytes correspond to the field layout in ORDER record and are extracted into the string ORD\$.

```

03530 LET JDATE=ASC(NEWORD$(33,36))

```

The Julian date is extracted from bytes 33 to 36 of NEWORDER.

```

03540 STMA 11,JDATE,MM,DD,YY
03550 LET DATE=YY*10000+MM*100+DD

```

A **STMA 11** is used to extract the year, month, and day components of the date and these are massaged into the format of ORDERS.ODATE

```

03600 LET ORD$(11,14)=CHR$(DATE,4)

```

and this is placed at the correct offset in the layout of the ORDERS record string

```

03610 LET MSGBUF$(32)=ORD$

```

which is placed in the message buffer string for submission to the U/SQL Server.

```

03620 RETURN

```

Here is an example of the results, using the UNIX Interactive U/SQL utility, [usqli](#):

```

U/SQL> insert into neworder (ordno, custcode, odate) values("123456",
"TEST", "1997-05-06");

```

```

U/SQL> select * from neworder where ordno="123456";

```

```

ORDNO  CUSTCODE      ODATE
-----
123456 TEST          1997-05-06

```

```

U/SQL> select * from orders where ordno="123456";

```

```

ORDNO  CUSTCODE      ODATE
-----
123456 TEST          970506

```

Performance of the U/SQL Record Mapper

There is obviously a performance overhead in using the U/SQL Record Mapper, in that each record read by the BBASIC ISAM Data Source Driver is first passed to the U/BL program that remaps it before it is passed back to the Data Source Driver.

The **NEWSTOCK** file, in the example above, with 5529 records gives a sample comparison of using and not using the U/SQL Record Mapper:

Using the UNIX Interactive U/SQL utility, [usqli](#), the following SQL queries were issued:

- `select sum(stkvalue) from NEWSTOCK;`

This query triggered the U/SQL Record Mapper and took 18 seconds to process.

- `select sum(price) from NEWSTOCK;`

This query did not trigger the U/SQL Record Mapper and only took 6 seconds to process.

Note: *The print statements were removed from the sample U/BL server program so that they would not influence the result.*

Retrieving the Record Number (R1) using SQLGetStmtOption

The following information is only useful if you are programming at the ODBC call level interface.

If a new row is added to a table defined on a data record, it would be useful to know what record number was given to the entry. This number can then be used for the insertion of the key. This information can be retrieved using the **SQLGetStmtOption** call. It has the following format:

```
RETCODE SQLGetStmtOption(hstmt, fOption, pvParam)
```

Where the arguments are as follows:

| | |
|---------|---|
| hstmt | Statement handle. Must be the same handle as used for the insert. |
| fOption | Set this to 1001. This is a Transoft escaped call. |
| pvParam | The record number is returned in this parameter. |

The following example, using Visual Basic code, shows how a data record and then a key may be inserted independently of each other. Locking will automatically take place to provide data integrity

```
` Insert data record
INSERT1 = "insert into tabl values ('CUST 1',2,4,6,0)"
SQLExecDirect(hstmt, INSERT1, len(INSERT1))
` Get the R1 of the inserted record
SQLGetStmtOption(hstmt, 1001, r1)
` Insert the key
INSERT2 = "insert into idx values ('CUST 1',2," + r1 + ")"
SQLExecDirect(hstmt, INSERT2, len(INSERT2))
```

Locking

In order to ensure that locking takes place between your ODBC-enabled application and your U/BL, UBB or B32 system, the directive **BBLockType** must be set.

On UNIX this is set in either the **[Data Source Defaults]** section or the UDD's **[<Data Source>]** section in the U/SQL Server **usqlsd.ini** configuration file. This is located in the **bin** directory, below the base directory where the U/SQL Server software was installed on the host, for example, **/usr/usqls/bin**.

On Windows this can be set using the [U/SQL Administrator](#) or the [U/SQL Service Manager](#).

BBLockType is defined as follows:

```
BBLockType=UBB|UBL|B32|NONE
```

The setting will depend on which product you are using.

- The default, if **BBLockType** is not set, is **UBB**.
- To ensure that no locking takes place, you must set the value to **NONE**.

The following is an example of the entries in the **usqlsd.ini** configuration file, including **BBLockType** for U/BL locking:

```
[company.udd]
Directory=/u/data01
BBLockType=UBL
ReadOnly=Yes
```

Locking takes place at the record level immediately prior to a record being updated or deleted via **UPDATE** and **DELETE** statements. However, the **SELECT FOR UPDATE** statement only locks the current record and not the cursor of records being updated. In order to ensure that all records are locked until the transaction is complete, your own application must make use of a Named Cursor.

For Multiple-tier U/SQL, set the server **Locktimeout=** directive to -1 (the default) to ensure that your application waits indefinitely on any lock set by any other process or ODBC application.

Setting Up a U/FOS Data Dictionary

Setting Up a U/FOS Data Dictionary

U/FOS is Transoft's Open Systems version of Data General's INFOS proprietary hierarchical database management system. This section describes setting up a UDD for U/FOS and applies only to Multiple-tier installations for both UNIX and Windows NT Server platforms.

Note: Refer to the [Overview](#) and [Creating a UDD](#) sections for an overview of the dictionary technology and the steps involved in creating a UDD.

Relational View of U/FOS Data

To provide a relational view of the data contained in a U/FOS database, you must go through a two-step process:

1. You must analyze the database and define one or more SQL tables, each of which represents a particular subset of the database.
2. You must define how each piece of data contained in the table is retrieved from the U/FOS database.

Each SQL table is capable of representing any combination of data contained in data records, keys or partial records. In addition, you can have things like the feedback of a data record or the occurrence number of a key. It is possible to create a single table that represents the entire U/FOS database. In general, though, there should be a separate SQL table for each type of data record. This table should also contain any key or partial record associated with the data record. Once the layout of each table has been decided, you must specify the key path needed to retrieve each piece of data contained in the table.

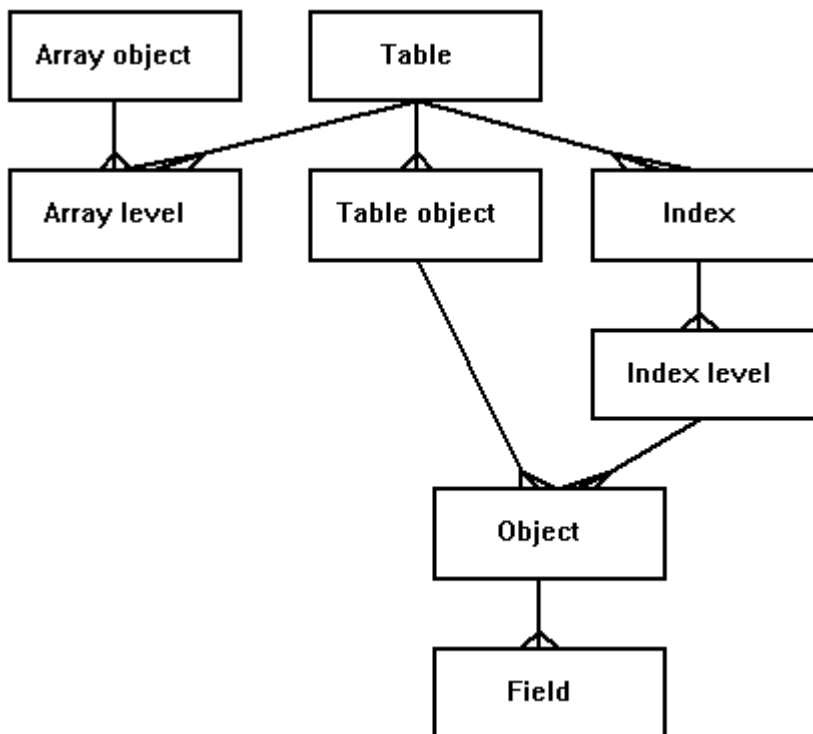
All this information is contained in the Universal File Dictionary (UFD). The Universal Data Dictionary (UDD) is automatically created from the UFD. The UDD contains details of the 'relational' tables and their column names that ODBC-enabled products will 'see'.

Universal File Dictionary Overview

The UFD is a collection of information contained in SQL tables which specifies the layout of each SQL 'relational' data table and where to retrieve the information. There are two main items that need to be described about each table: its data objects and its indexes.

Each table is made up of one or more data objects where a data object is either data from a data record, partial record, key or a special object type: occurrence, feedback, link or subscript. Each object will be made up of one or more fields. This information is sufficient to completely describe all the columns within the table. These objects are discussed in the [Data Objects](#) section.

Each table will have one or more indexes attached to it. Each of these indexes will be made up of one or more sub-index levels.



The following table contains a brief description of the contents of each table within the UFD:

| | |
|--|---|
| <u>ufos table</u> | Contains the name of each data table, the number of components and the number of indexes. |
| <u>ufos tab object</u> | Contains the name of each data object for each data table. This name is used to extract further information about the object from the ufos_object table. |
| <u>ufos index</u> | Contains the name of the index, the number of levels, the name of the file containing the index and its pathid. |
| <u>ufd pathnames</u> | Stores the paths to the physical database files and is used by the ufos_index table. It contains the pathname and the pathid. |

| | |
|--|--|
| <u>ufos_index_level</u> | Contains information about each level of each index. There are two types of information for each level. The first element of information concerns the components that may be retrieved from that level. The second element describes the attributes of the sub-index level, such as whether or not sub-indexes are allowed and whether duplicates are allowed. |
| <u>ufos_object</u> | Contains information that applies to each data object: the name, number of fields and type. |
| <u>ufos_field</u> | Contains information about each field in each component. The information stored about each field is: field name, the COBOL picture string, the usage and the value of it, if constant. |
| <u>ufos_array_object</u> | Only used for arrays that are contained in separate tables from the parent table. The information contained is the name of each array and the level of nesting. |
| <u>ufos_array_level</u> | Contains information about each level of each array declared in the ufos_array_object table. The information stored is: <ul style="list-style-type: none"> • the name of the table containing the array • the start offset • the size of each element • the number of elements. |

Review of the UFD Tables

This section reviews each of the UFD tables in turn:

- [ufos_table](#)
- [ufos_tab_object](#)
- [ufos_index](#)
- [ufd_pathnames](#)
- [ufos_index_level](#)
- [ufos_object](#)
- [ufos_field](#)
- [ufos_array_object](#)
- [ufos_array_level](#).

There is an example in the section, [Example Data Dictionary](#).

ufos_table

This table contains data that applies to each table as a whole:

ufos_table

| | | |
|-------------|------------|---|
| tablename | char (18) | Name of the table. |
| nind | smallint | Number of indexes attached to table. |
| nobj | smallint | Number of data objects in table |
| read_direct | char (1) | 'Y' if reading direct. |
| cond | char (200) | Conditional test to see if a record belongs to table. |

tablename is the name of the table that ODBC-enabled products will 'see'. Table names must be SQL compliant and satisfy the following conditions:

- are in UPPERCASE
- are within 18 characters
- are unique
- are not SQL keywords
- are not zero length
- are made up of 0-9, A-Z, _, with the first character alphabetic and with no blanks.

If the **read_direct** field is set, the index structure is bypassed and records are retrieved directly, which is a lot quicker. This does, of course, mean that all the records in the database are retrieved for each sequential read.

The **cond** field is used to set a condition under which a data record belongs to the specified table. The syntax of the condition is identical to that used for an SQL

WHERE clause search-condition (see the [Expression Handling](#) section). Set the condition field if either of the following apply:

- If the records are being read directly and there are several types of record in the database.
- If there several types of record in the same sub-index. See the [Multiple Record Types under the same Sub-Index](#) section.

ufos_tab_object

This table is simply a list of the data objects in a specified table and the **objno** field describes the ordering.

ufos_tab_object

| | | |
|-----------|-----------|--|
| tablename | char (18) | Name of the table to which the object belongs. |
| objno | smallint | The sequence number of the object. |
| object | char(16) | The name of the object. |

The **object** field is used to retrieve further information about the object from other tables.

ufos_index

This table contains information that applies to each index as a whole.

ufos_index

| | | |
|-----------|-----------|---|
| tablename | char (18) | Table to which index belongs. |
| indno | smallint | Index identification number. |
| indname | char(16) | The name of the index. |
| nlevel | smallint | The number of sub-index levels that makes up the index. |
| stop_lev | smallint | The sub-index level at which to stop searching for further records. |
| filename | char(33) | The name of the physical file containing the index. |
| pathid | smallint | The directory path number to the physical file. The actual path is supplied in the ufd_pathnames table. |

nlevel contains the total number of sub-index levels contained in the index.

stop_lev is used when the same record type is contained in multiple sub-indexes. The U/SQL Server U/FOS data source driver recursively backtracks up

sub-index levels searching for further records. The number in this field indicates that the search must not go beyond this level. If records are not kept in multiple sub-indexes specify the first level (0).

It is recommended, for greater flexibility, that you set **pathid** to zero so that the filename will be searched for down the **Searchlist=** entry in the U/SQL Server directives rather than in the **ufd_pathnames** file; see the [ODBC.INI Directives](#) section for further details.

ufd_pathnames

This table stores the paths to the physical database files and is used by the [ufos_index](#) table, although as indicated above use of the **Searchlist=** directive is preferable.

ufd_pathnames

| | | |
|----------|-----------|--|
| pathname | char(254) | The directory path to the physical database files. |
| pathid | smallint | The key number for the directory path. |

UNIX

On UNIX platforms, each directory path entry can be one of the following:

- The full path name to the directory where the file is located, for example, **/usr/datafiles**.
- The directory relative to the base directory where the data files are located (for instance, **/usr/datafiles**). For example, suppose a particular data file is in the directory **/usr/datafiles/sales**, then its relative path to the base data files directory is **sales**.

Note: If the directory entries are of this type, there must be a '**Directory=/usr/datafiles**' entry in the U/SQL Server configuration file, [usqlsd.ini](#).

- The directory relative to where the U/SQL Server software resides. For example, assume the U/SQL Server is started in **/usr/usqls/bin** and the data file is in **/usr/datafiles**, the relative path is **../../datafiles**.

Note: If the directory entries are of this type, there must be a '**Directory=../**' entry in the U/SQL Server configuration file, [usqlsd.ini](#).

Windows NT Server

Enter equivalent Windows NT Server path names, to those for UNIX, for example, **C:\DATAFILES**.

ufos_index_level

This table describes which objects may be found at each level of an index.

ufos_index_level

Transoft U/SQL User Guide

| | | |
|----------------|-----------|---|
| indname | char (16) | Name of index to which this level belongs. |
| levno | smallint | The sequence number of the level. |
| key_obj | char(16) | The name of the key object to be found at this level. |
| data_obj | char(16) | The name of the data record object to be found at this level, if any. |
| partial_obj | char(16) | The name of the partial record object to be found at this level, if any. |
| occurrence_obj | char(16) | The name of the occurrence number object to be found at this level, if any. |
| feedback_obj | char(16) | The name of the object to be found at this level, if any. |
| allow_subidx | char(1) | 'Y' if sub-indexes are allowed. |
| allow_dup | char(1) | 'Y' if duplicates are allowed. |
| allow_partial | char(1) | 'Y' if partial records are allowed. |
| max_key_len | smallint | Maximum key length; 0 if default. |
| first_key | char(16) | For system use only. |
| last_key | char(16) | For system use only. |

There will always be a key object, **key_obj**, but the other objects can be left unspecified if they do not exist or are not used. The other fields specify the attributes of the sub-index, such as the maximum key length and whether or not duplicates are allowed.

ufos_object

This table contains information that applies to each object as a whole.

ufos_object

| | | |
|----------|-----------|-----------------------------|
| objname | char (16) | Name of the object. |
| numfield | smallint | Number of fields in object. |
| objtype | char(16) | The type of the object. |

Valid object types in **objname** are KEY, PARTIAL and DATA which describe data stored in keys, partial records and data records. There are also the following special object types: OCCURRENCE, FEEDBACK, LINK and SUBSCRIPT. These objects are described in the [Data Objects](#) section.

ufos_field

This table contains information about each field.

ufos_field

| | | |
|---------|-----------|--|
| objname | char (16) | Name of the object that this field is a part of. |
| fldno | smallint | Sequence number of the field. |
| fldname | char (30) | Name of field. |
| picture | char (18) | COBOL picture expression for field. |
| fusage | char (14) | Usage of field. |
| fvalue | char (18) | Value for field. This is only used for fields that are in selector objects. |
| offset | char (4) | Field offset for redefined fields only. If the field is not a redefine, offset should be left blank. |

fldname is the column (field) name that ODBC-enabled products will 'see'. Column names must be SQL compliant and satisfy the following conditions:

- are in UPPERCASE
- are within 18 characters
- are unique
- are not SQL keywords
- are not zero length
- are made up of 0-9, A-Z, _, with the first character alphabetic and with no blanks.

If the name of the field **fldname** is FILLER, it will not appear in the UDD table.

Valid types for the field **fusage** are: DISPLAY, STRING, COMP, COMP-3 and INDEX. This has been enhanced to introduce new data types, these are described in [Appendix C - U/FOS data type enhancements](#).

The **fvalue** field is only set if the field is part of a selector object.

ufos_array_object

This table is only used when an array is defined as a separate logical table to its parent record:

ufos_array_object

| | | |
|--------|-----------|--------------------------|
| name | char (18) | Name of array object. |
| nlevel | smallint | Number of nested levels. |

There is one entry in this table for each array object, where an array object is either a simple single-level array or a nested array.

ufos_array_level

This table contains information about each level of an array.

ufos_array_level

| | | |
|--------------|-----------|--|
| objname | char (18) | Name of array object to which this level belongs. |
| tablename | char(18) | The name of the table to which this array level belongs. |
| levelno | smallint | The sequence number of the level. |
| start_offset | smallint | The byte offset of the start of the array level. |
| element_size | smallint | The size of an element in the array. |
| max_elements | smallint | The maximum size of the array. |

The **tablename** field contains the name of the table that this array is attached to. Further information can be found in the [Array Handling](#) section.

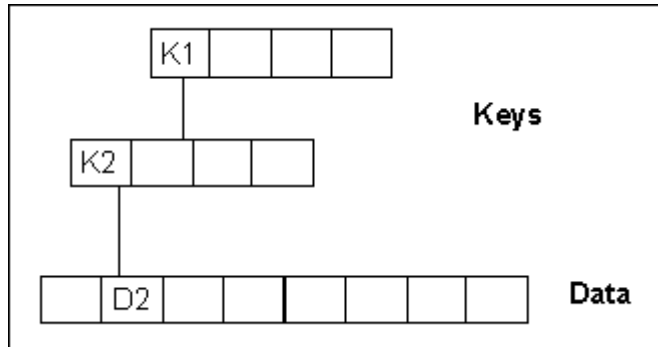
Data Objects

All 'relational' SQL tables viewed by ODBC-enabled products are constructed from one or more data objects. Each object represents a piece of data either contained in the database itself or generated by the U/SQL Server U/FOS data source driver. The following seven data object types are supported:

- **KEY**: this object represents a U/FOS key.
- **SELECTOR**: this object is the same as a key object but contains data that is constant. This is used for selectors.
- **DATA**: this object contains the data in a data record.
- **PARTIAL**: this object contains the data in a partial record.
- **OCCURRENCE**: this object contains the occurrence number of a key.
- **FEEDBACK**: this object contains the feedback of a data record.
- **SUBSCRIPT**: this object contains the subscript of the current element of an array.

Indexes

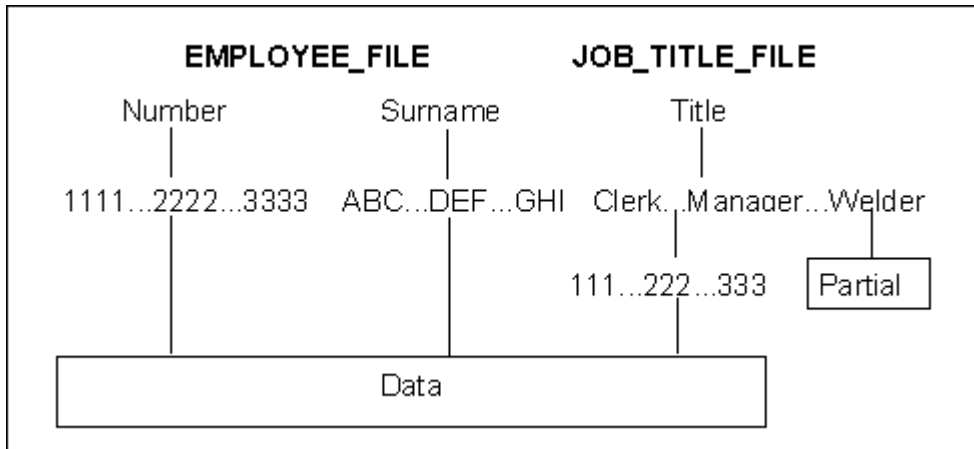
An index attached to an SQL table is a composite of all the keys on the key path to the data record or partial record. All the keys on this key path must be contained in the table, otherwise you would not be able to find the location of the data record. For example, assume you have an SQL table defined for the data record in the following structure:



This table would have a single composite index made up of K1 and K2. If the information contained in the keys was not duplicated in the data record, you would also have to have K1 and K2 as objects in the table.

Example Data Dictionary

You will now work through the creation of a data dictionary that represents the following example database:



The data record has the following format, represented by the COBOL FD:

```
01 EMPLOYEE-REC.
   02 NUMBER      PIC 9(4)    COMP.
   02 SURNAME     PIC X(20) .
   02 ADDRESS     PIC X(30) .
   02 PHONE       PIC 9(10) .
```

The partial record has the following format:

```
01 JOB-TITLE-REC.
   02 MIN-PAY     PIC 999     COMP-3.
   02 MAX-PAY     PIC 999     COMP-3.
```

Let's now see what the entries for each UFD table should be in the '.ufd' text file:

ufos_table (tabname,nind,nobj,read_direct,cond)

| tabname | nind | nobj | read_direct | cond |
|-----------|------|------|-------------|------|
| EMPLOYEE | 3 | 2 | | |
| JOB_TITLE | 1 | 2 | | |

The table EMPLOYEE represents the EMPLOYEE-REC data record and the table JOB_TITLE represents the JOB-TITLE-REC partial record.

The EMPLOYEE table is composed of two objects: the EMPLOYEE-REC data record and the TITLE key. The TITLE key object is included because all key information must be present in the table and the job title is not in the data record. The table has three indexes: Number, Surname and Title + Number.

The JOB_TITLE table is composed of two objects: the JOB-TITLE-REC partial record and the TITLE key. The TITLE key object is included because all key information must be present in the table. The table has one index: Title.

It is not possible to read these records directly as you have no way of differentiating between a data record and a partial record.

Note: The **tabname** names must be SQL compliant, see the previous section, **ufos_table**. For example, the field separator character '-' is illegal in SQL syntax. Use '_' instead.

The following table defines a name for each object in each table.

Transoft U/SQL User Guide

ufos_tab_object (tabname, objno, object)

```

-----
EMPLOYEE      1      EMPREC
EMPLOYEE      2      TLEKEY
JOB_TITLE     1      TLEKEY2
JOB_TITLE     2      JOBREC
*
```

The following table gives a name to each index, defines the number of sub-index levels and specifies the name of the file containing the index. **nlevel** is set to zero since you do not have multiple instances of the same sub-index type. Also included is the **pathid**.

ufos_index (tabname, indno, indname, nlevel, stop_lev, filename, pathid)

```

-----
EMPLOYEE      1  NUMIDX  2  0  EMPLOYEE_FILE  1
EMPLOYEE      2  NAMIDX  2  0  EMPLOYEE_FILE  1
EMPLOYEE      3  TLEIDX  3  0  JOB_TITLE_FILE 1
JOB_TITLE     1  TLEIDX2 2  0  JOB_TITLE_FILE 1
*
```

The next table denotes the UNIX directory path where the database files may be found, with the **pathid** number. Preferably use the **Searchlist=** directive, which means setting **pathid** to zero in the **ufos_index** table.

ufd_pathnames (pathname, pathid)

```

-----
/usr/usqls/data_files      1
*
```

The next table specifies which objects may be found at each level of each index. From the above table you can see that duplicate surnames are allowed.

ufos_index_level (indname, levno, key_obj, data_obj, partial_obj, occurrence_obj, feedback_obj, allow_subidx, allow_dup, allow_partial, max_key_len)

```

-----
NUMIDX  1  NUMSEL
NUMIDX  2  NUMKEY  EMPREC

NAMIDX  1  NAMSEL
NAMIDX  2  NAMKEY  EMPREC          Y

TLEIDX  1  TLESEL
TLEIDX  2  TLEKEY
TLEIDX  3  NUMKEY2  EMPREC

TLIDX2  1  TLESEL
TLIDX2  2  TLEKEY2          JOBREC          Y
*
```

The next table defines the type of each object and the number of fields contained in it.

ufos_object (objname, numfield, objtype)

```

-----
NUMSEL      1      SELECTOR
NAMSEL      1      SELECTOR
NUMKEY      1      KEY
NUMKEY2     1      KEY
NAMKEY      1      KEY
TLEKEY      1      KEY
TLEKEY2     1      KEY
```

```

EMPREC          4      DATA
JOBREC          3      PARTIAL
*
```

The data object types, **objtype**, supported are detailed in the [Data Objects](#) section .

The following table describes each field for each object. Notice that the fields for the selector objects have the name as FILLER. This means that the selectors will not appear as columns in the table when viewed by ODBC-enabled products. Also notice that the value has been set for the selector fields.

ufos_field (objname, fldno, fldname, picture, fusage, fvalue)

| objname | fldno | fldname | picture | usage | fvalue |
|---------|-------|---------|---------|---------|---------|
| NUMSEL | 1 | FILLER | X(6) | DISPLAY | NUMBER |
| NAMSEL | 1 | FILLER | X(7) | DISPLAY | SURNAME |
| NUMKEY | 1 | NUMBER | 9(4) | COMP | |
| NUMKEY2 | 1 | NUMBER | 9(4) | COMP | |
| NAMKEY | 1 | SURNAME | X(20) | DISPLAY | |
| TLEKEY | 1 | TITLE | X(10) | DISPLAY | |
| TLEKEY2 | 1 | TITLE | X(10) | DISPLAY | |
| EMPREC | 1 | NUMBER | 9(4) | COMP | |
| EMPREC | 2 | SURNAME | X(20) | DISPLAY | |
| EMPREC | 3 | ADDRESS | X(30) | DISPLAY | |
| EMPREC | 4 | PHONE | 9(10) | DISPLAY | |
| JOBREC | 1 | MIN_PAY | 999 | COMP-3 | |
| JOBREC | 2 | MAX_PAY | 999 | COMP-3 | |

*

Array Handling

Note: A detailed discussion on array handling is provided in the [Handling Data Arrays](#) section.

When dealing with arrays, there are two choices: expand the array out into multiple columns or define the array as a separate logical table. For small arrays, it is probably best to expand them out, but large arrays must be put in a separate table. Let's now look at an example of how to define the array as a separate table.

```
01 EXAMPLE-REC.
    03 CODE-KEY          PIC 99.
    03 NAME              PIC X(20) OCCURS 10.
    03 PREV-ADDR        OCCURS 9 TIMES.
        05 FLAT          PIC X(10).
        05 ROADS         PIC X(10) OCCURS 3 TIMES.
        05 TOWN          PIC X(10).
```

The above COBOL FD contains one single level array and a two-level nested array. From this record, you want to be able to create three tables: a table containing all elements of the NAME array, a table containing all elements of the PREV-ADDR array and a table containing all elements of the ROADS array.

The first stage is to create entries in the various UFD table for all three data tables:

ufos_table (tabname, nind, nobj, read_direct, cond)

```
-----
NAME          1      2
ADDR          1      2
ROAD         1      2
*
```

ufos_tab_object (tabname, objno, object)

```
-----
NAME          1      NAME_REC
NAME          2      NAME_SUB
ADDR          1      ADDRREC
ADDR          2      ADDR_SUB
ROAD          1      ROADREC
ROAD          2      ROAD_SUB
*
```

ufos_index (tabname, indno, indname, nlevel, stop_lev, filename, pathid)

```
-----
NAME          1  NAMEIDX  1  0  EXAMPLE_FILE
ADDR          1  ADDRIDX  1  0  EXAMPLE_FILE
ROAD          1  ROADIDX  1  0  EXAMPLE_FILE
*
```

ufos_index_level (indname, levno, key_obj, data_obj, partial_obj, occurrence_obj, feedback_obj, allow_subidx, allow_dup, allow_partial, max_key_len)

```
-----
NAMEIDX  1  NAME_KEY  NAMEREC
ADDRIDX  1  ADDR_KEY  ADDRREC
ROADIDX  1  ROAD_KEY  ROADREC
*
```

There must also be a column to identify the element number of the array. This is achieved by adding an object to the **ufos_object** table of type: SUBSCRIPT. This object must contain a field for each level of nesting. In the example, the SUBSCRIPT object for the NAME and ADDR tables would each contain one field, whereas the object for the ROAD table would contain two fields. The fields, in the **ufos_field** table, for the subscript objects must be numeric, must be long enough to contain the highest subscript and must also have **fusage** DISPLAY.

ufos_object (objname, numfield, objtype)

```

-----
NAME_KEY      1      KEY
ADDR_KEY      1      KEY
ROAD_KEY      1      KEY
NAMEREC      3      DATA
ADDRREC      6      DATA
ROADREC      4      DATA
NAME_SUB      1      SUBSCRIPT
ADDR_SUB      1      SUBSCRIPT
ROAD_SUB      2      SUBSCRIPT

```

*

When you come to defining the fields in the data record, though, there should only be fields defined for the first element of the array: the rest of the elements should be described as a FILLER field, which are not 'seen' by ODBC-enabled products. Also, the key data item must be present in all the tables so that you have a field to join them on. The entries for the data records in **ufos_field** table are as shown below:

ufos_field (objname, fldno, fldname, picture, fusage, fvalue)

```

-----
NAME_KEY 1      NAME      X(20)  DISPLAY
NAMEREC  1      CODE_KEY  99     DISPLAY
NAMEREC  2      NAME      X(20)  DISPLAY
NAMEREC  3      FILLER    X(180) DISPLAY
NAME_SUB  1      SUB1      9(4)   DISPLAY
ADDR_KEY  1      ADDR      X(30)  DISPLAY
ADDRREC  1      CODE_KEY  99     DISPLAY
ADDRREC  2      FILLER    X(200) DISPLAY
ADDRREC  3      FLAT      X(10)  DISPLAY
ADDRREC  4      FILLER    X(30)  DISPLAY
ADDRREC  5      TOWN      X(10)  DISPLAY
ADDRREC  6      FILLER    X(400) DISPLAY
ADDR_SUB  1      SUB1      9(4)   DISPLAY
ROAD_KEY  1      ROAD      X(10)  DISPLAY
ROADREC  1      CODE_KEY  99     DISPLAY
ROADREC  2      FILLER    X(210) DISPLAY
ROADREC  3      ROAD      X(10)  DISPLAY
ROADREC  4      FILLER    X(410) DISPLAY
ROAD_SUB  1      SUB1      9(4)   DISPLAY
ROAD_SUB  2      SUB2      9(4)   DISPLAY

```

*

The next stage is to add entries to the **ufos_array_object** and **ufos_array_level** tables. The **ufos_array_object** table contains an entry for

each array object, where an array object is either an elementary array or an array with one or more levels of nesting. The column **nlevel** describes the level of nesting. For the above example, the table is as follows:

ufos_array_object (name,nlevel)

```

-----
NAME          1
ADDR          2
*
```

The **ufos_array_level** table describes each individual array within an array object. The column **table_name** describes the name of the table to which the array belongs. The rest of the columns describe the position and length of the array. For the above example, the table is as follows:

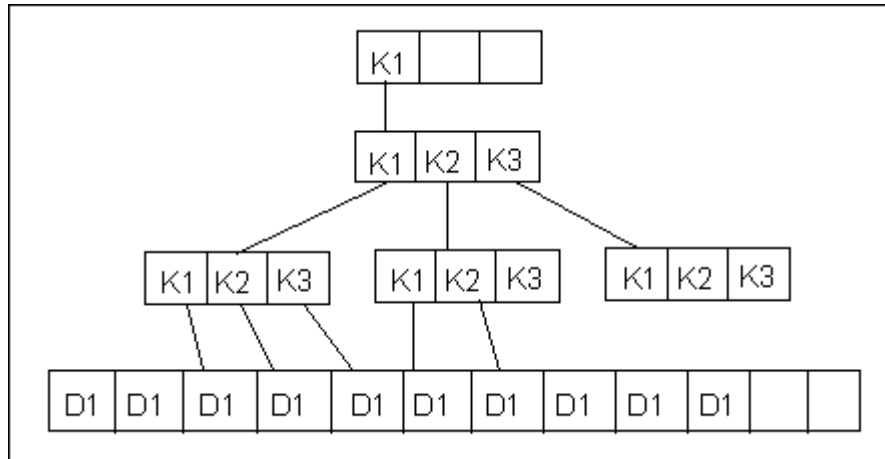
ufos_array_level (obj_name, table_name, level, start_offset, element_size ,max_elements)

```

-----
NAME      NAME      1  2   20  10
ADDR      ADDR      1 202  50   9
ADDR      ROADS     2 212  10   3
*
```

Same record type Under Multiple Subindexes

In some databases, there will be records of the same type that are under several sub-indexes. Take the following example structure:



In this case, you really want to view the bottom level sub-indexes as being joined to form one big sub-index. The way you achieve this is by setting the **stop_lev** field in the **ufos_index** table. The U/SQL Server U/FOS data source driver first reads the data records under the bottom left-most sub-index. It then backtracks and goes down the next sub-index.

It appears that the driver automatically performs what is required in this case. The only problem is that, after the last bottom-level sub-index has been read, the driver attempts to backtrack up to the first level and go down the next key path. You do not want the driver to do this, as you know that there will be not be any more data records of the required type. The way to stop this is to set the **stop_lev** field to the sub-index level that you do not want to backtrack past. The first level is level 0 so, in this case, set **stop_lev** to 1.

Multiple Record Types Under the Same Sub-index

In some databases, there will be records of different types that are under the same sub-index. You would obviously represent each record type as a separate logical table. The problem is how to know which record belongs to which table. To solve this problem, there must be one or more fields which you can examine that will tell you the record type. You can then specify an expression under which a record belongs to a table. This expression or condition is placed in the **cond** field of the table **ufos_table** and has the same grammar as an SQL WHERE clause search-condition. See the [Expression Handling](#) section.

For example, suppose there are two records, Employee Master and Employee Salary, under the same sub-index. They are distinguished from each other by the REC_TYPE field being "M" and "S" respectively.

Thus, the cond expression would be:

```
REC_TYPE="M"
```

for the Employee Master table and

```
REC_TYPE="S"
```

for the Employee Salary table.

The **ufos_table** table entries would be as follows:

```
ufos_table (tabname,nind,nobj,read_direct,cond)
-----  --  --  --  -----
MASTER      1      10  REC_TYPE="M"
SALARY      1      6   REC_TYPE="S"
*
```


Enterprise Join Engine

Enterprise Join Engine

The **Transoft Enterprise Join Engine (EJE)** creates a logical data source which contains links to a number of physical data sources, either U/SQL or third-party data sources, typically RDBMS, using an ODBC connection. This enables distributed data, over multiple data sources, and/or multiple data sets of the same data, for example, multiple companies as U/SQL enabled data, to be presented as one logical database. This enables multiple U/SQL data sources and/or multiple, disparate, non-U/SQL, ODBC-enabled data sources (such as Informix, Oracle, SQL Server 7, and so on) to be joined together and queried as one data source.

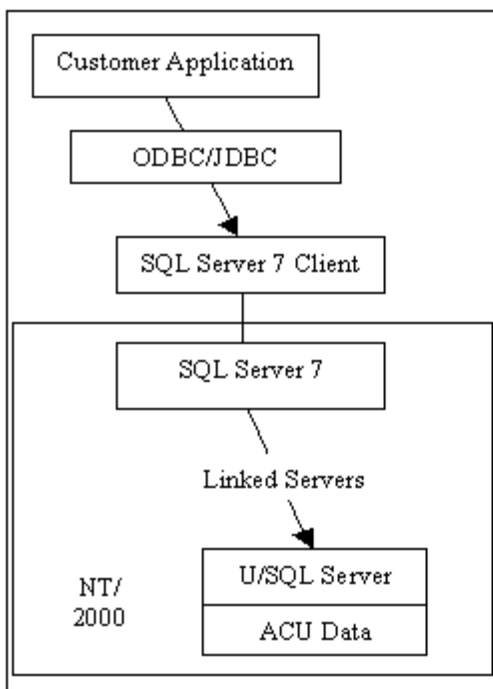


Figure 1. Driven by SQL Server 7

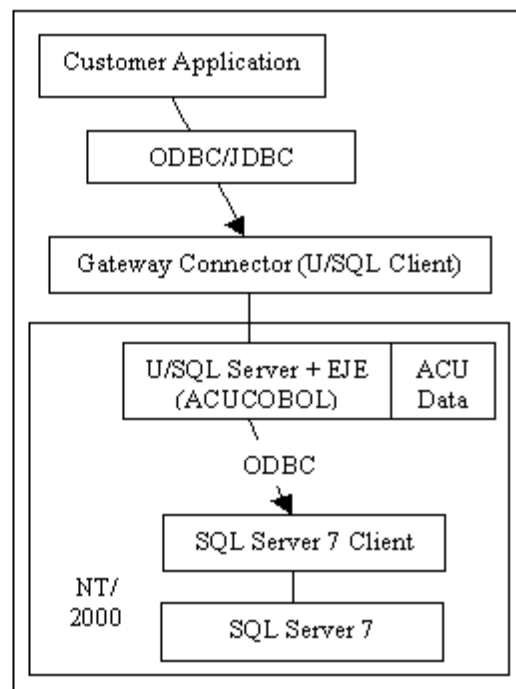


Figure 2. Driven by U/SQL EJE

The EJE was introduced to tackle scenarios where customers wished to join data in SQL Server 7 and U/SQL (see **Figure 1**). However, this had several problems, notably that it was slow due to SQL Server 7 not asking the remote data sources about indices, and that due to difficulties in the way the Linked Servers mechanism worked it proved impossible for Transoft to support this interface for anything other than a very limited read-only capacity.

Using the EJE (as shown in **Figure 2**) turns this scenario around and makes U/SQL rather than SQL Server 7 the driving force, allowing Transoft control over the link and therefore being able to solve the issues associated with **Figure 1**. The EJE in conjunction with U/SQL is now responsible for optimising SQL queries (leading to significantly improved results). This ensures full index use where available, and allows a supported read-write interface, seamlessly joining the two data sets (in this case ACUCOBOL U/SQL and SQL Server 7) in a reliable and efficient fashion.

A key point of **Figure 2** (and therefore the EJE in general) is that wherever data is to be joined together between the two (or more) data sets, the EJE must be the entry point into the system, allowing it to drive the queries.

Note: *Many data sources can be joined together as one logical data source. **Figure 2** is designed to give a simple introduction to possible EJE configurations. There is no limit to the number of data sources which may be joined.*

Other uses of the EJE include:

- **Data pump**

The data pump capability of the EJE allows data to be pumped from one database to another using SQL commands (thus, enabling the conditional selection of data to be transferred), and without the target database having to have its own "data pump" capability.

- **Multi-company consolidation**

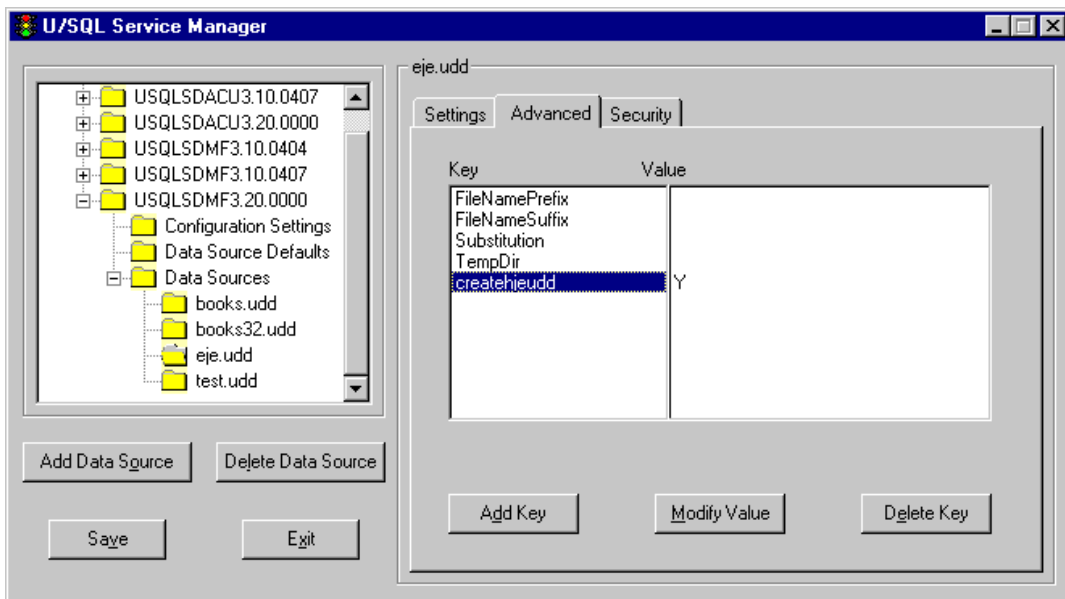
If a customer has multiple UDDs (Universal Data Dictionaries) or multiple DSN configurations to the same UDD (for example, multiple substitution strings), all running under U/SQL, the EJE can be used to join across multiple companies. For example, a customer list could be compared between two companies to check which customers exist in both, or totals of combined business/accounts could be compiled as if the database were one entity.

Using the Enterprise Join Engine

Preparing to use the EJE

To use the Transoft Enterprise Join Engine take the following steps:

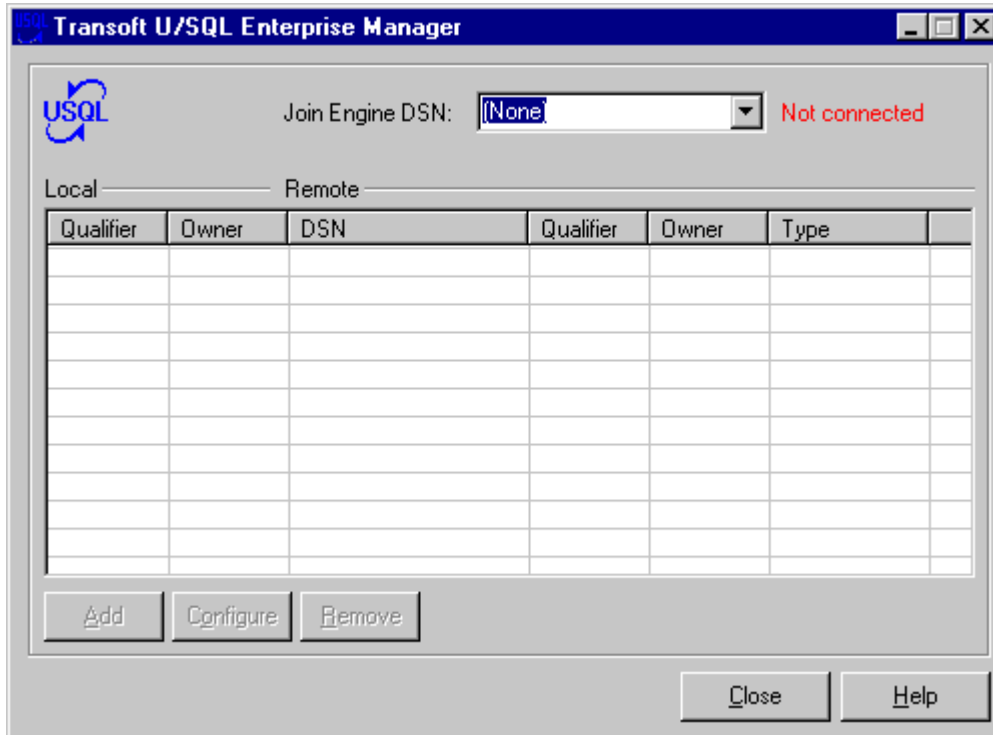
1. Using the [U/SQL Service Manager](#) create a Data Source entry for your EJE Data Dictionary. See [Creating a Data Source entry using the U/SQL Service Manager](#) section.
2. Specify the directive "**createhjeudd=y**" for your EJE data source:



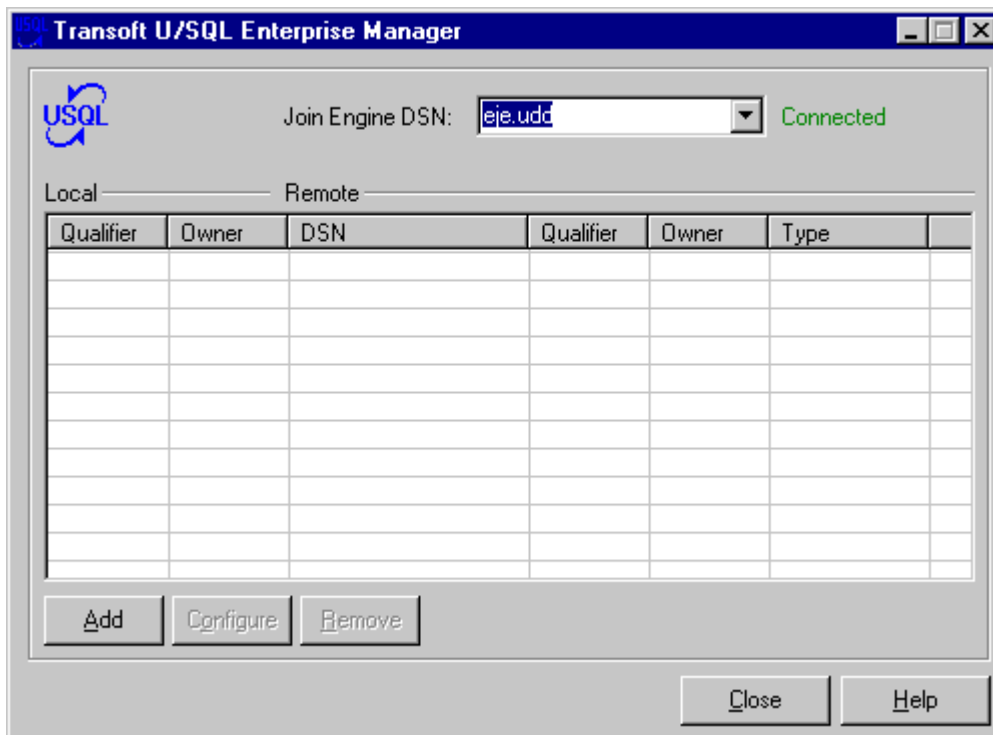
3. Create a remote EJE UDD, called for example, **EJE.UDD**. This will have two new system UDD tables over and above the standard. See the [Creating a UDD](#) section.
4. Invoke the Transoft U/SQL Enterprise Manager.

U/SQL Enterprise Manager

When you invoke the Transoft U/SQL Enterprise Manager, the **Transoft U/SQL Enterprise Manager** property sheet is displayed:



Specify the **EJE.UDD** from the pull down menu in the **Join Engine DSN** field, and the Enterprise manager will connect to it:

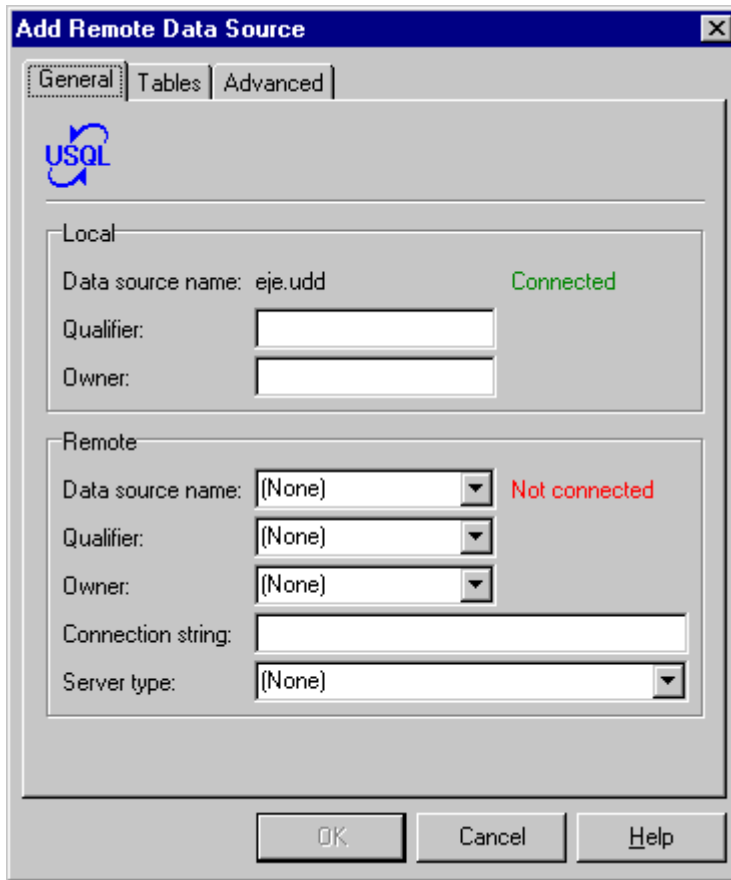


This property sheet allows you to:

- [Add a Remote Data Source](#)
- [Configure a Remote Data Source](#)
- [Delete a Remote Data Source.](#)

Add a Remote Data Source

To add a remote data source click **Add**. The **Add Remote Data Source** property sheet is displayed:



This property sheet contains three property pages:

- [General](#)
- [Tables](#)
- [Advanced](#).

General

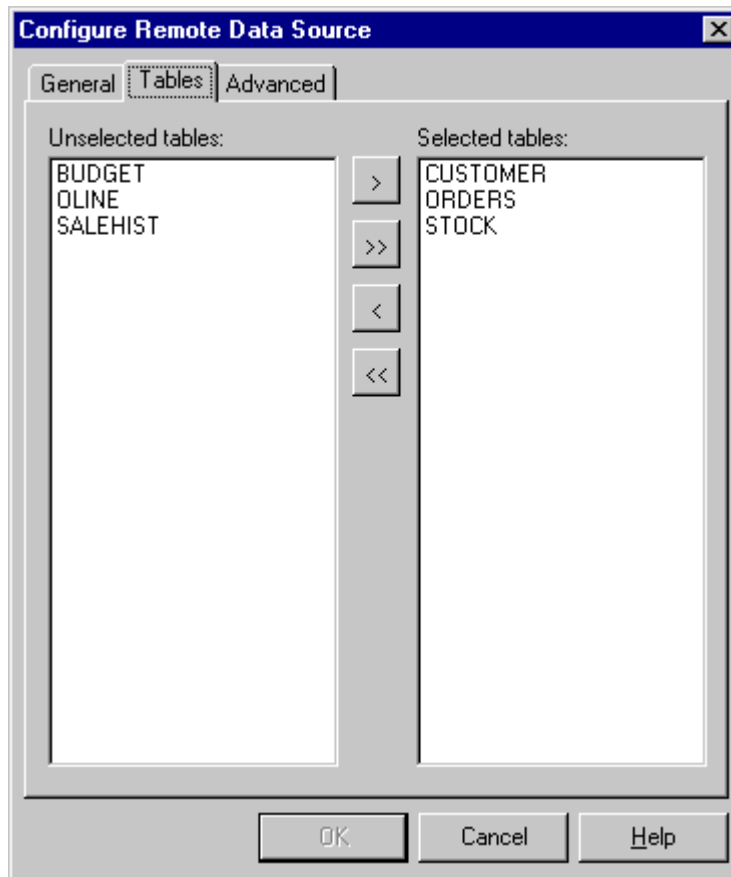
By default the **General** property page is displayed. This **General** property page contains the following fields:

| Local | |
|-------------------------|---|
| Data source name | This is the data source name of the EJE server. In the above example, it is "eje.udd". |
| Qualifier | This mandatory field represents the qualifier section of the name of this remote EJE data source as it will appear when connecting to the EJE. For example, if Qualifier is set to "myqual" and Owner is set to "myowner" and a table "customer" is imported (see Tables section below), the table can be referenced as "myqual.myowner.customer". If there is only one "customer" table it may still be referenced |

| | |
|--------------------------|--|
| | <p>as "customer".</p> <p>This field cannot be amended after a Remote Data Source has been added.</p> |
| Owner | <p>This mandatory field represents the owner section of the name of this remote data source. See Qualifier above for details.</p> <p>This field cannot be amended after a Remote Data Source has been added.</p> |
| Remote | |
| Data source name | <p>This is the remote data source to be connected into the EJE.</p> <p>This field cannot be amended after a Remote Data Source has been added.</p> |
| Qualifier | <p>This drop-down list provides a list of the qualifiers within the remote data source. For U/SQL data sources it should be left as "(None)". It is only required in the case where the remote data source also splits its tables into multiple qualifiers. See note in Tables section below.</p> <p>This field cannot be amended after a Remote Data Source has been added.</p> |
| Owner | <p>This drop-down list provides a list of the qualifiers within the remote data source. For U/SQL data sources it should be left as "(None)". It is only required in the case where the remote data source also splits its tables into multiple owners. See note in Tables section below.</p> <p>This field cannot be amended after a Remote Data Source has been added.</p> |
| Connection string | <p>This field contains the connection string which was used to connect to the Remote Data Source. Bear in mind that this connection string represents that used from the local machine, that is, where the Enterprise Manager is running, not necessarily where the EJE is running. It may be amended as appropriate to connect to the Remote Data Source.</p> |
| Server type | <p>This drop-down list provides a list of the types of remote data sources the EJE server is configured to work with. This affects what properties the remote server is assumed to have and how the SQL syntax can be treated. For example, some remote data sources do not allow "SELECT FOR UPDATE" queries.</p> |

Tables

Click the **Tables** tab. The **Tables** property page is displayed:



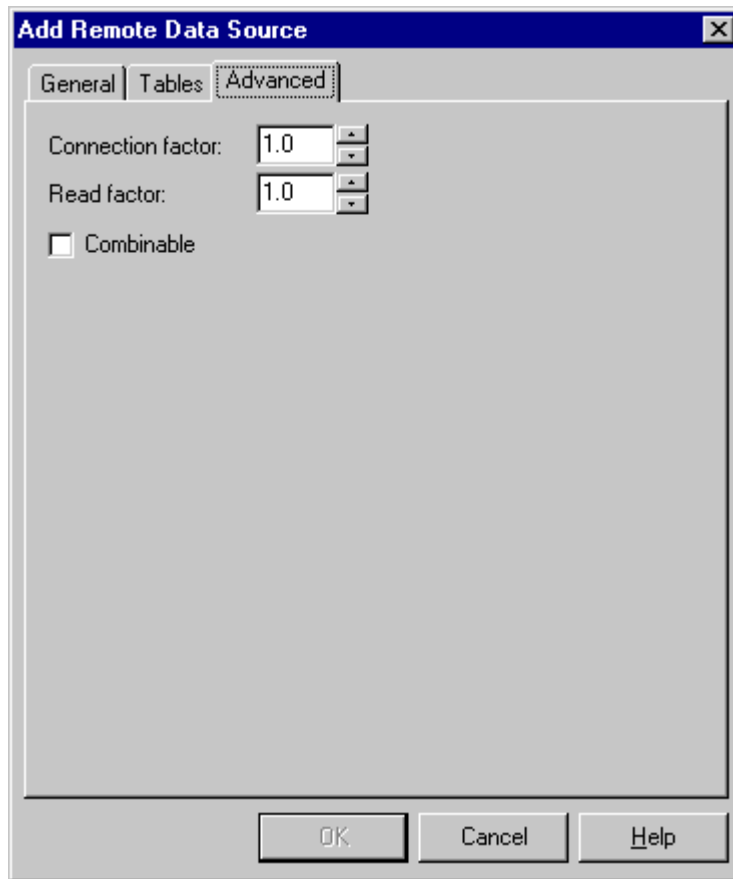
This property page allows you to specify which tables you want to exclude from your remote data source.

Note: The list of tables provided in this tab is based on the qualifier and owner selected under the remote section on the [General](#) property page. If the remote data source actually divides its tables into owners and/or qualifiers, then this list will change depending on what is selected there. Always ensure that the list of tables is as expected before clicking **OK**.

Advanced

Note: In normal operation it is recommended that you do not change any of these fields.

Click the **Advanced** tab. The **Advanced** property page is displayed:



This property page contains the following fields:

| | |
|--------------------------|---|
| Connection factor | This allows tuning of performance and states how much slower preparing a remote statement is than on the central EJE server. |
| Read factor | This allows tuning of performance and states how much slower reading a row of data is on the remote server than on the central EJE server. |
| Combinable | This check box is used when multiple remote data sources are connected, which actually correspond to a single physical remote data source, just with multiple remote qualifiers and/or owners. For example, two remote data sources could link to a single SQL Server 7 database, one with a remote owner of "a1" and one with a remote owner of "a2" and a query combining the two could be joined as in "select * from a1.table1, a2.table2". The Combinable flag means that anything with the same qualifier will be the same remote data source, and only one connection will be opened to it. This flag is not required unless remote qualifiers and owners are used, which is not normally the case. |

Configuring a Remote Data Source

To configure a remote data source, select the data source and click the **Configure** button on the U/SQL Enterprise Manager. The **Add Remote Data Source** property sheet for that data source is displayed. This property sheet contains the property pages and parameters described in the previous section [Add a Remote Data Source](#).

Deleting a Remote Data Source

To delete a remote data source, select the data source and click the **Remove** button on the U/SQL Enterprise Manager. A confirmation prompt is displayed. Click **Yes** to delete the data source.

Configuring the EJE on UNIX

The following steps allow you to configure and use the EJE on UNIX:

1. U/SQL version v4.00 is the earliest U/SQL release to support the EJE. If you are not already running this version then you must first install this release on your UNIX machine.
2. The U/SQL v4.00 Server must be licensed with an **FMD66** (EJE enabled) license that also specifies an SQL Access (**UD**) count.
3. In your U/SQL Server installation **bin** directory, edit the **usqlsd.ini** file, and add the following directive to the [Data Source Defaults] section.

```
createhjeudd=y
```

The EJE depends on a series of specific **UDDTABLE** entries and the directive above ensures they are created whenever a new UDD is created.

4. The U/SQL UNIX 'client' (**libtsodbc**) requires an **ODBC.INI** file so that remote datasources can be located and identified. In the U/SQL **bin** directory, create an **ODBC.INI** file and add your remote DSN entries.

For example, if you were connecting to another U/SQL Server on a different UNIX machine, or in a different directory, then you would add the relevant port and hostname details.

```
[eje1]
```

```
Server= <your server hostname>
```

```
Port=< server port number>
```

5. The EJE must also be able to determine what UNIX 'client driver' to use. This is specified in an **iodbc.ini** file. If you were connecting to another U/SQL DSN you would enter the following in the **iodbc.ini** file:

```
[ODBC Data Sources]
```

```
books.udd=Transoft Data Source
```

```
eje1=Transoft Data Source
```

```
[books.udd]
```

```
Driver=< U/SQL server install directory path  
>/sqlaccess/libtsodbc.so
```

```
[eje1]
```

```
Driver=< U/ SQL server install directory path  
>/sqlaccess/libtsodbc.so
```

```
[default]
```

```
Driver=< U/SQL server install directory path  
>/sqlaccess/libtsodbc.so
```

6. In order for the U/SQL Server to be able to locate the various configuration elements and the U/SQL 'UNIX Client' you must set the following environment variables PRIOR to starting the U/SQL Server on UNIX:

```
export ODBC_INI=`pwd`/ODBC.INI
```

```
export ODBCINI=`pwd`/iodbc.ini
```

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:< U/SQL server install  
base directory path >/iodbc
```

7. Start the U/SQL Server process:

```
./start_serv.sh
```

- The next step is to create the **EJE.udd** using the standard **-c** switch (using **usqli**):

```
./usqli -c eje.udd
```

This data dictionary will 'house' all of your EJE remote tables and serves as a single reference point.

- Before you populate your **EJE.udd** with remote table information you must add an **EJE.udd** DSN reference in your U/SQL Client Administrator.

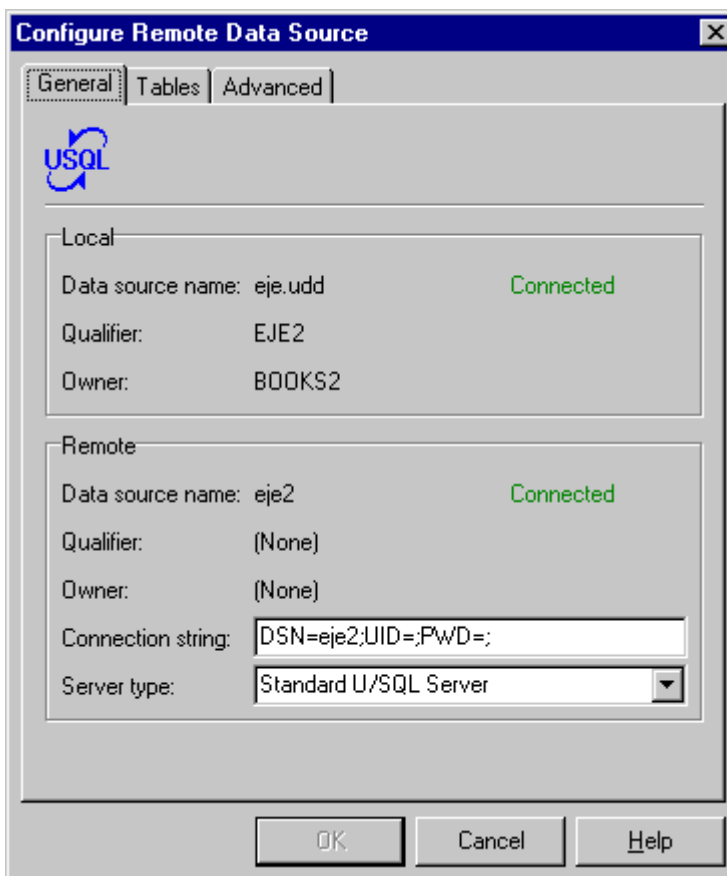
The **EJE.udd** DSN entry must be created as a 'System' DSN.

- After adding the U/SQL Administrator entry for the **EJE.udd** you can begin the process of populating it with your remote table information.

Version 4.00 of the U/SQL Client installs the EJE Administrator within the U/SQL program group (if you cannot locate it search for the **EJEAdmin.exe**.)

To begin adding information, open the EJEAdmin and connect to the **eje.udd**.

Click **ADD** and complete the fields as in the example below:



The **Tables** tab allows you to select which remote tables you wish to reference in your **EJE.UDD**.

There should be no need, unless specifically instructed, to amend the Advanced settings.

- You can test whether you can connect to your EJE datasource by connecting to your **eje.udd** in [Win U/SQLi](#) or [usqli](#) and running:

```
select top 1 * from eje2.books2.customer
```

Confirm the results (note the use of the Qualifier, Owner and tables name in the SQL Syntax).

Configuring Multi-Datasource U/SQL EJE on UNIX

This section describes a series of steps that will allow you to configure and use the EJE to co-join disparate data using two U/SQL Servers.

It describes:

- The basic steps involved in using the EJE on UNIX
- Configuring EJE to connect to multiple U/SQL Servers on UNIX.

The test scenario was based on:

- Two U\SQL Servers on a single UNIX machine
- C-ISAM and PRO-ISAM datasources.
- The **books.udd** (and associated test data).

To configure and use the EJE to co-join disparate data using two U/SQL Servers, take the following steps:

1. Install the U/SQL Version 4.00 PRO-ISAM release in a UNIX directory called, **pro400**.
2. Install The U/SQL Version 4.00 C-ISAM release in a UNIX directory called, **cisam400**.
3. When installed License both servers with with an EJE enabled license (**FMD66**) and ensure each license has an SQL Access count (**UD**) count specified.
4. In the USQL PRO400 **BIN** directory, edit the **usqlsd.ini** file and add the following directive in the [Data Source Defaults] section:

```
createhjeudd=y <<<<---- add this
```

Also add the following new section within the file:

```
[eje1]
Dictionary=../example/books.udd
Directory=../example
```

5. In the **CISAM400** directory, edit the **usqlsd.ini** file and add the following EJE DSN entry within the [Data Source Defaults] section:

```
[eje2]
Dictionary=../example/books.udd
Directory=../example
```

6. Create a file in the U/SQL PRO400 **bin** directory called **ODBC.INI**. Edit the file and add the following:

```
[eje1]
Server= <your server hostname>
Port=< PROISAM server port number>
[eje2]
Server=<your server hostname>
Port=< CISAM server port number>
```

7. Create a file in the U/SQL PRO400 **bin** directory called **iodbc.ini**. Edit the file and add the following:

```
[ODBC Data Sources]
books.udd=Transoft Data Source
eje1=Transoft Data Source
eje2=Transoft Data Source
[books.udd]
Driver=< PROISAM server install directory path
>/sqlaccess/libtsodbc.so
[eje1]
Driver=< PROISAM server install directory path
>/sqlaccess/libtsodbc.so
[eje2]
Driver=< PROISAM server install directory path
>/sqlaccess/libtsodbc.so
[default]
Driver=< PROISAM server install directory path
>/sqlaccess/libtsodbc.so
```

8. Return to the U/SQL PRO400 **BIN** directory and then run the following:

```
export ODBC_INI=`pwd`/ODBC.INI
export ODBCINI=`pwd`/iodbc.ini
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:< PROISAM server install
directory
path >/iodbc
```

9. Start the PRO400 and CISAM400 server processes.

10. In the PRO400 **bin** directory, run:

```
./usqli -c eje.udd
```

This will create the **eje.udd** that will 'house' all remote table references.

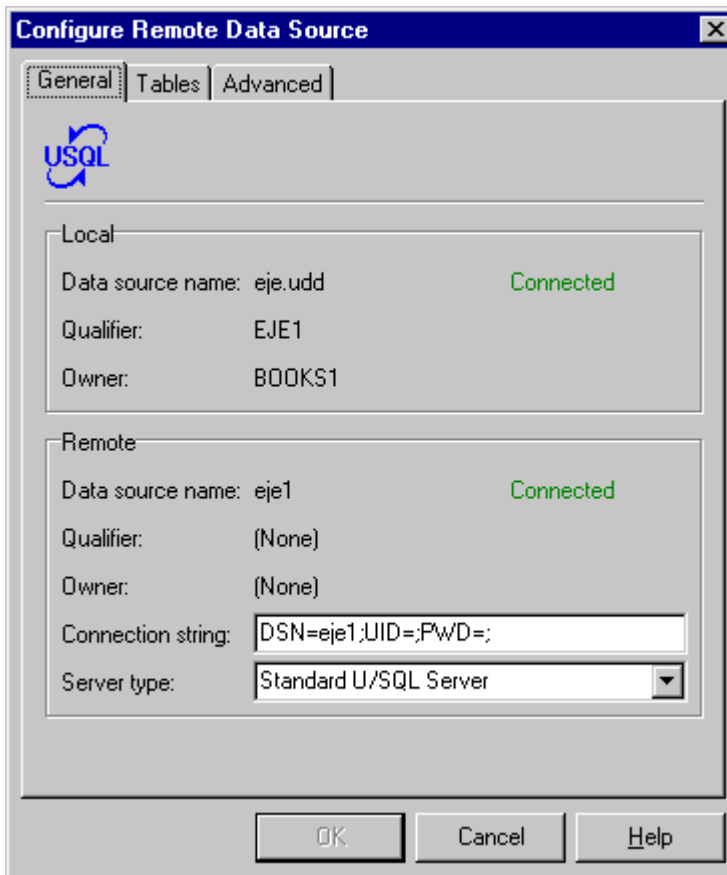
11. In your USQL Client Administrator create System DSN's for:

eje.udd server = MyServer, port = < proisam server port number>

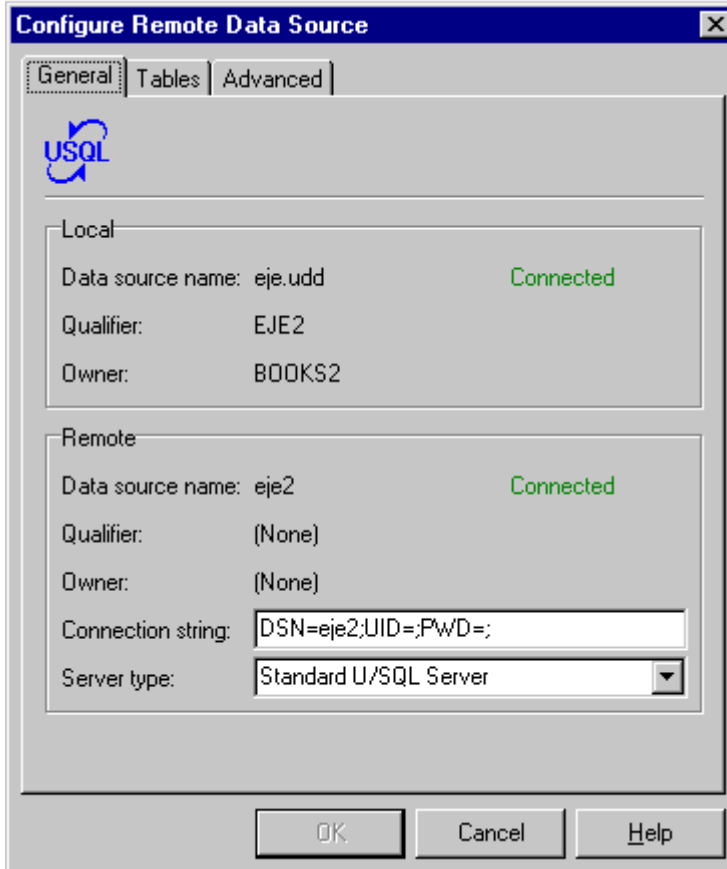
eje1 (do NOT specify .udd extensions) server = MyServer , port = < proisam server port number>

eje2 server = MyServer, port = < cisam server port number>

12. Open the **EJAdmin** (accessed from the U/SQL Client v4.00 program group) and connect to the **eje.udd**. Click **ADD** and complete the fields as in the example below:



ADD another reference as in the further example below:



13. Connect to your **eje.udd** in **Winusqli** and run:

```
select top 1 * from ejel.books1.customer  
and then  
select top 1 * from eje2.books2.customer  
and confirm the results.
```

Configuring U/SQL EJE Thin Client to Connect to the SQL Server

This example was based on a UNIX based U/SQL Server connecting to SQL Server 7 on a Windows client machine. In addition, the U/SQL Server software was also installed on the same machine as SQL Server 7 to act as the 'Thin Client'.

For test purposes the **books.udd** CUSTOMER table was used as the primary U/SQL and SQL Server table example.

1. Install the **UNIX U/SQL version 4.00** server software on your UNIX Server.
2. Install the **U/SQL Server version 4.00.0200** on your Windows (NT/2000/XP Pro) client PC.
3. License both servers with theEJE enabled (**FMD66**) license with a (**UD**) **SQLACCESS** Count.
4. In the U/SQL UNIX **bin** directory, edit the **usqlsd.ini** file, and add the following to the [Data Source Defaults] section:

```
createhjeudd=y
```

5. Create a file in the U/SQL UNIX **bin** directory called, **ODBC.INI**. Edit the file and enter the following:

```
[books.udd] <<<<<<<< This is your UNIX based DSN
Server=sgsco506
Port=7000
[pro4] <<<<<<<<<<<<<<<<<<<<< This is your Windows based SQL Server DSN
Server=aleahy <<<< This is the name of the windows machine hosting
U/SQL / SQL Server
Port=7001 <<<<<<<< This is the port number on which the U/SQL
Windows Server is running
```

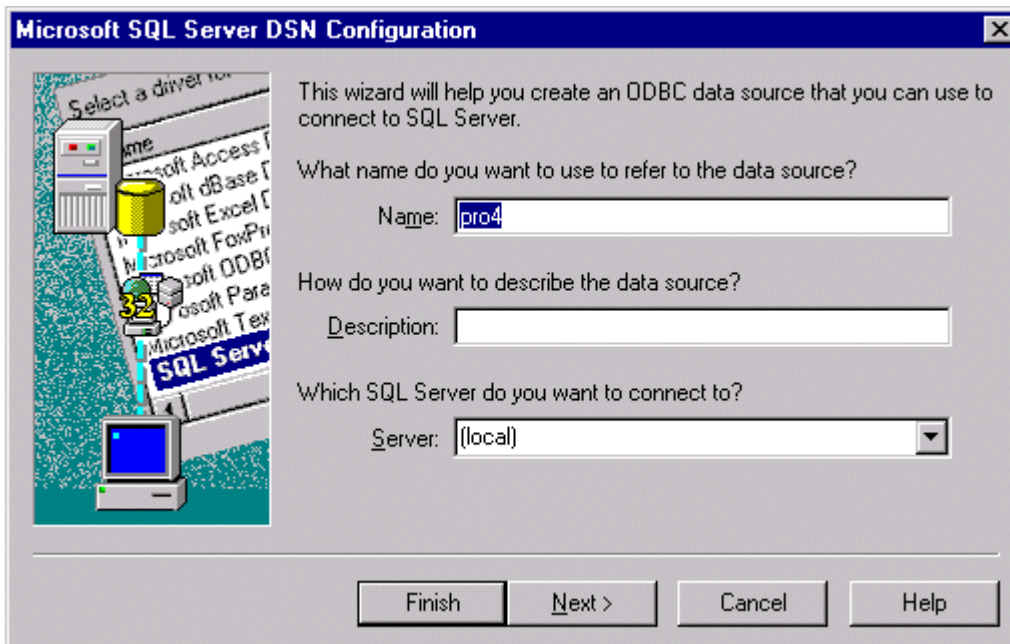
6. Create a file in the U/SQL UNIX **bin** directory, called, **iodbc.ini**. Edit the file and enter the following:

```
ODBC Data Sources]
books.udd=Transoft Data Source
pro4=Transoft Data Source
[books.udd]
Driver=/home/proj/support/aleahy/pro400/sqlaccess/libtsodbc.so
[pro4]
Driver=/home/proj/support/aleahy/pro400/sqlaccess/libtsodbc.so
[default]
Driver=/home/proj/support/aleahy/pro400/sqlaccess/libtsodbc.so
```

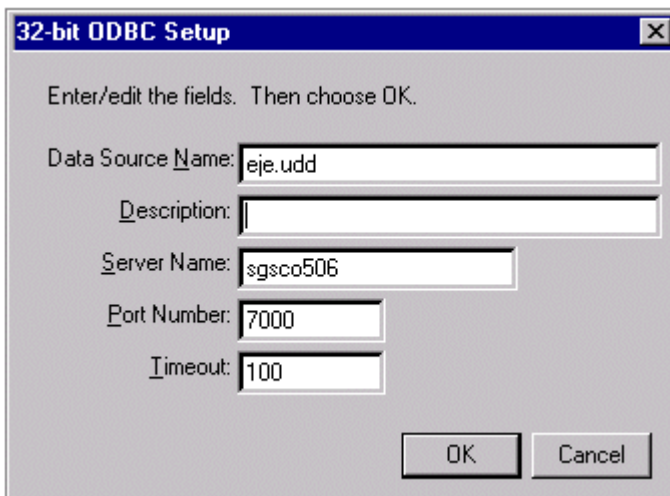
7. In the U/SQL UNIX **bin** directory run:

```
export ODBC_INI=`pwd`/ODBC.INI
export ODBCINI=`pwd`/iodbc.ini
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<your_usql_base_directory_path>/i
odbc
```


8. Start the U/SQL UNIX Server and create the **EJE.udd**:
`./usqli -c eje.udd`
9. On your client computer create a System DSN for your SQL Server Database in the Microsoft ODBC Administrator:

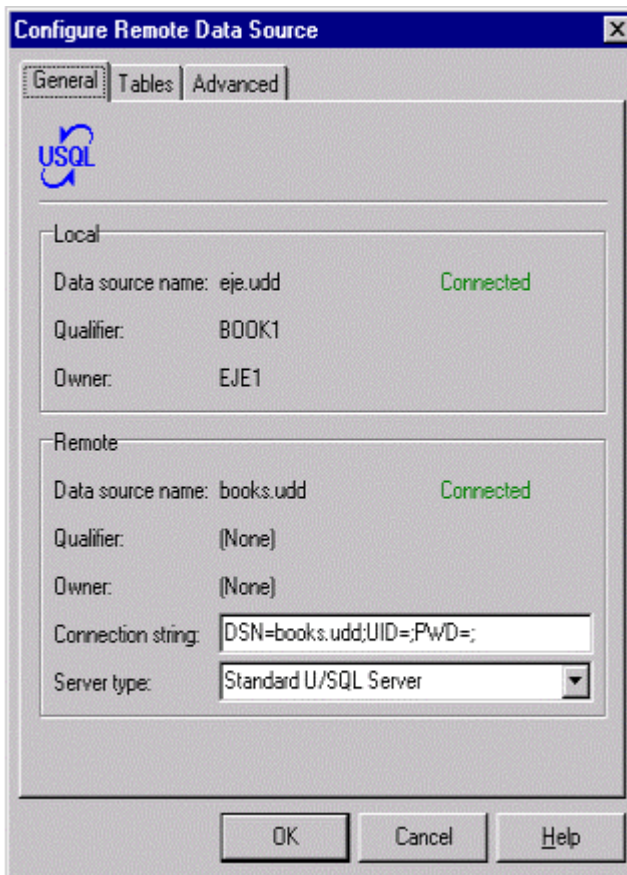


On your client machine create a System DSN for your **EJE.udd** in the U/SQL Administrator:

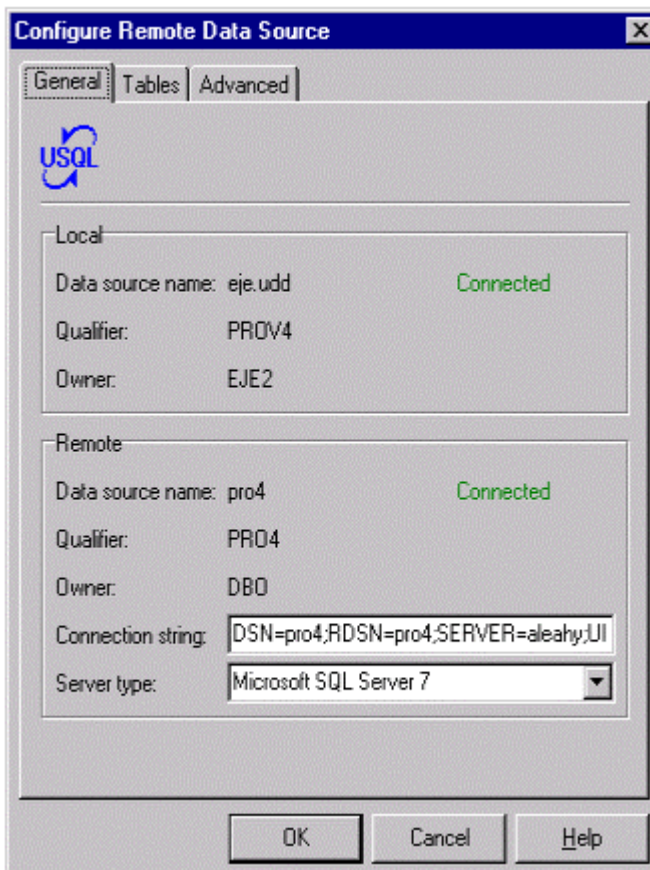


10. Open your EJE Administrator on your client machine and populate with DSN entries for your U/SQL and SQL Server datasources:

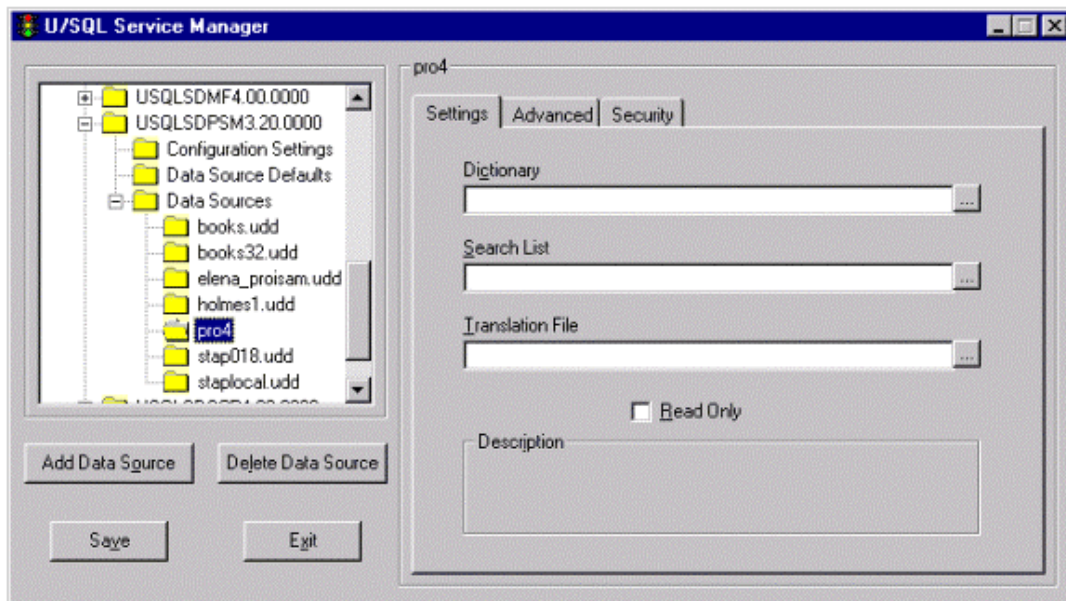
U/SQL DSN



SQL Server DSN (manually edit connect string & add RDSN=<sql_server_dsn> & edit SERVER=sql_servername)



11. In the U/SQL Service Manager expand the **Service** entry and add a DSN entry that mirrors your SQL Server ODBC DSN:



Start the U/SQL Service.

12. Using **USQLi** on UNIX, connect to the **eje.udd** and test the datasources:

```
$ ./usqli eje.udd
Interactive U/SQL Utility.
Copyright (c) Transoft Ltd. 1993-2003
Connected to server: [Engine v4.00.0103][PRO-ISAM v4.00.0000]
Opened: 'eje.udd'
U/SQL> select top 3 * from book1.eje1.customer;
CUSTCODE CUST_NAME                                CUST_REGION
-----
C001     Alden & Blackwell                            OST
C002     Book Bargains Oxford                            WEST
C003     The Bookcentre                                 WEST
3 records retrieved
U/SQL> select top 3 * from prov4.eje2.customer;
CUSTCODE CUST_NAME                                CUST_REGION
-----
C001     Sein und Zeit                                  OST
C002     Book Bargains Oxford                            WEST
C003     The Bookcentre                                 WEST
3 records retrieved
```

Configuring U/SQL EJE for MS Access, MS Query or Crystal Reports

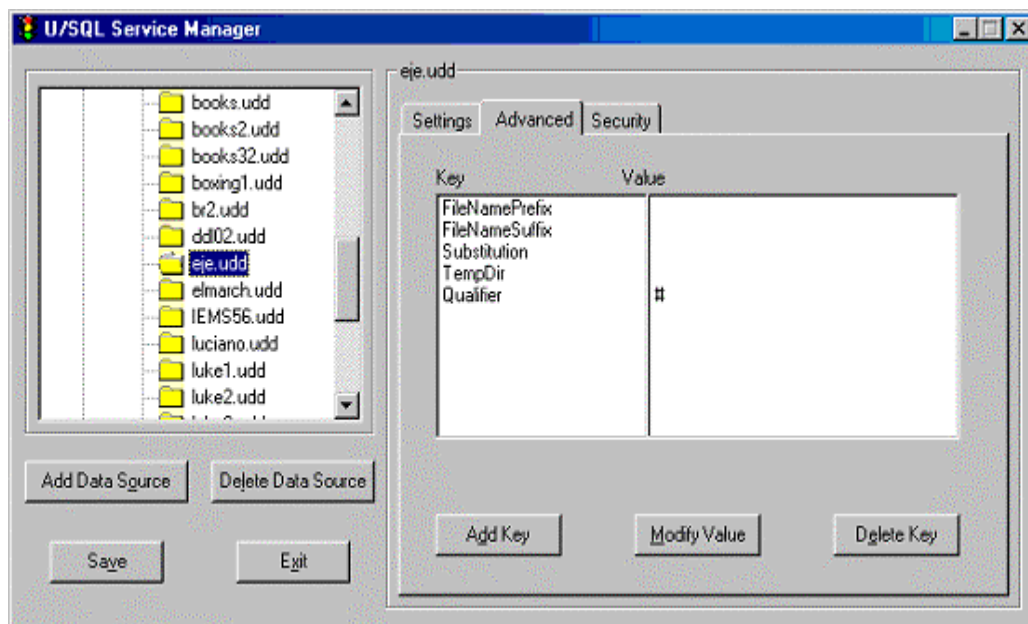
To Link/Import tables from the EJE.udd in MS Access, MS Query or Crystal Reports set Qualifier=# datasource defaults directive specific to the EJE.udd.

Qualifier=#

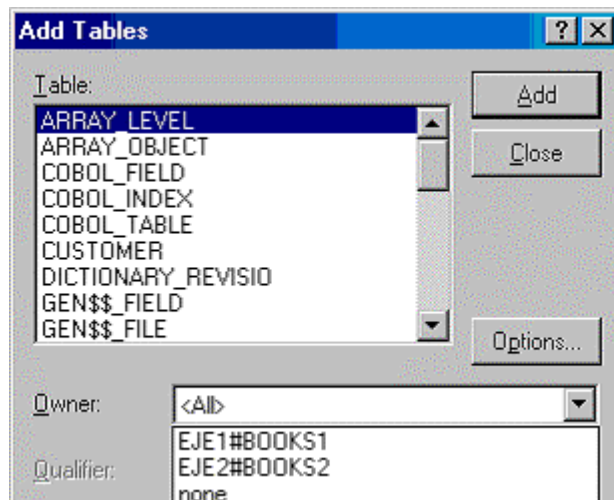
If set all table owners will be produced by concatenating qualifier and owner with the contents of this string. So if qualifier is set to # then a table q1.a1.customer can be referenced as q1#a1.customer, and will appear in SQLTables() results as being qualifier=NULL, author="q1#a1", tablename="customer".

Some front end applications have difficulty interpreting the connection string and this directive addresses that.

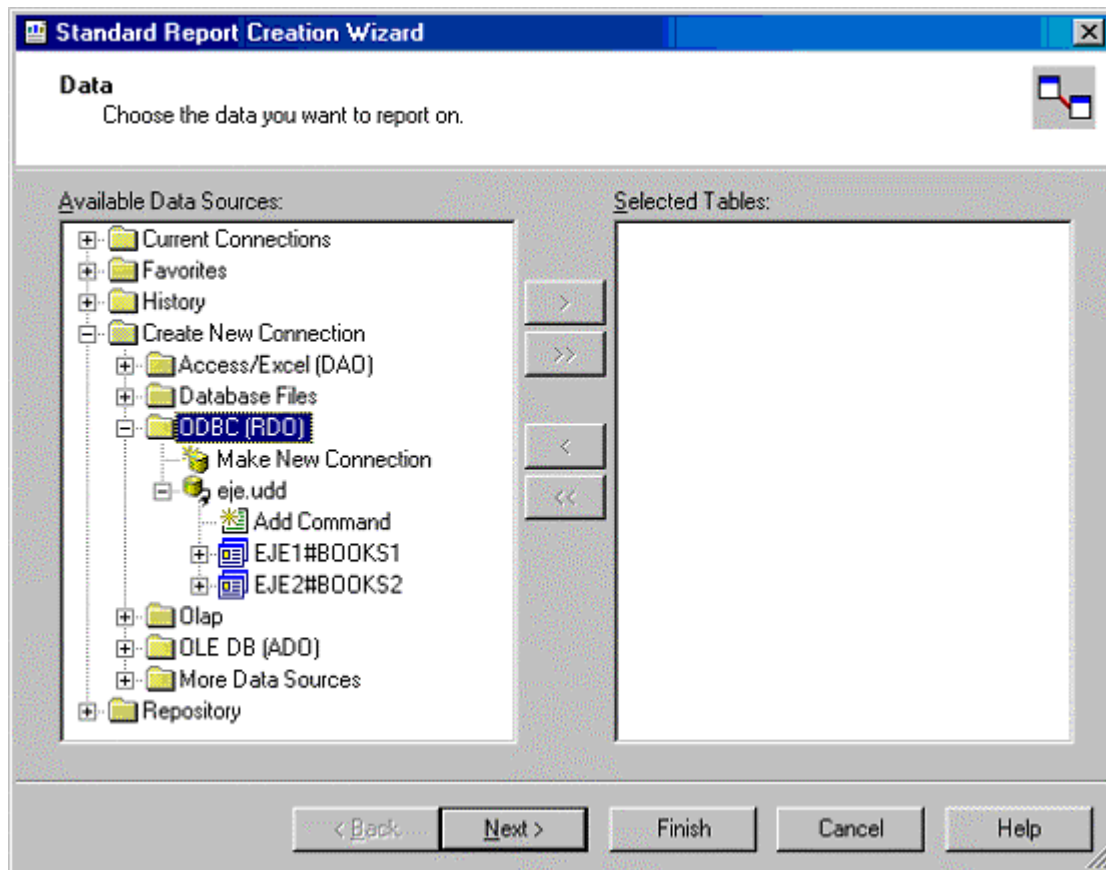
Set the Directive in U/SQL Service Manager:



This will overcome the issue in MS Query (and Access) where tables are not visible.



And in Crystal Reports as well.



Advanced Use

Advanced Use of U/SQL Adapters

This section describes more advanced use and features of U/SQL, including:

- [Multi-company Support](#)
- [Accessing the UDD system tables](#) - This is useful in establishing the available application tables and their columns.
- [Hints and tips](#)

Multi-company Support

U/SQL supports the concept of multi-company sets of data. For example, you may have two sets of data that refer to Company01 and Company02. The layouts of all the data files are the same, but some of the file names have differing prefixes or suffixes, for example, LEDGER01 and LEDGER02 are the main ledger files for the companies. In addition, the data files are in differing directories.

The UDD for the two companies would be the same, but the entry for the name of the main ledger file, in the UDD, would be, say, 'LEDGER##'. In the **ODBC.INI** settings there would be two data source entries, for example 'Company01' and 'Company02' with the same UDD but with differing file name substitution entries, ensuring that the ledger file '##' characters are substituted by '01' and '02' respectively. There would be a specific search list for each company's data files.

Consider the following example **ODBC.INI** settings:

```
[Company01]
Driver=C:\WINDOWS\SYSTEM\TSENG32.DLL
Description=Company 01
Dictionary=C:\Program Files\USQLC\COMPANY.UDD
DSDrivers=TSMF32
SearchList=F:\COMP01\
Substitution=##=01
```

```
[Company02]
Driver=C:\WINDOWS\SYSTEM\TSENG32.DLL
Description=Company 02
Dictionary=C:\Program Files\USQLC\COMPANY.UDD
DSDrivers=TSMF32
SearchList=F:\COMP02\
Substitution=##=02
```

Note: *In this example, instead of using the Substitution option, FileNameSuffix="01" and "02" respectively could be used to obtain the same physical file name to open. But in this case, the file name in the UDD would be 'LEDGER' not 'LEDGER##'.*

Issues and Limits

This section contains issues and restrictions that must be considered.

Note: Refer to the *U/SQL Client software Release Notice*, for any relaxation of these limits or other issues.

- **Creating dictionaries from UFD text files** - All Table names (for all data sources) and Column names (for non-COBOL data sources) must be in UPPERCASE

When creating the text file to define the UFD from which the UDD is subsequently created ensure that all table and column names are in UPPERCASE, otherwise unpredictable results may occur. This applies to all non-COBOL data sources, see the relevant chapters in the *Creating a Non-COBOL Dictionary* manual.

For COBOL data sources, only the table names must be in UPPERCASE. Column names may be mixed case. Refer to the *Setting Up a COBOL Data Dictionary* book.

Note: Non-COBOL column names are up to 18 characters, while COBOL column names may be optionally up to 30 characters. Ensure your ODBC-enabled product(s) can cope with column names greater than the SQL standard of 18 characters.

- **Check all UDD index entries**

It is extremely important to check that entries for each index contain all the key components in the correct order. Incorrect indexes will result in unpredictable results and performance degradation. Refer to the [Accessing UDD System Tables](#) section on how to interrogate the system index table describing your application's indexes and their key components.

- **Improve the access performance of 'old' dictionaries**

The access performance of dictionaries, created prior to revision 2.66 of U/SQL, may be improved by being recreated. For non-COBOL data sources use your existing UFD text file as described in the section [Creating the UDD from the UFD Text File](#).

For COBOL data sources you can export your existing UDD to a text file and then import it into a new UDD as described in the section [Exporting and Importing your UDD](#).

- **Dictionary limits**

A UDD can contain at least 2048 tables and each table can contain at least 512 columns. The largest SQL compliant character data type for a column is 254 bytes (this is an ODBC limitation). Alphanumeric fields larger than 254 must be 'split' into multiple columns. This is automatically performed by the U/SQL Manager for COBOL data sources.

- **Locking with SELECT FOR UPDATE statement**

U/SQL provides explicit locking at the record level immediately prior to a record being updated or deleted using **UPDATE** and **DELETE** statements. However, the **SELECT FOR UPDATE** statement only locks the current record and not the cursor of records being updated. In order to ensure that all records are locked until the transaction is complete, your own application must make use of a Named Cursor.

For Multiple-tier U/SQL, set the server **LockTimeout=** directive to -1 (the default) to ensure that your application waits indefinitely on any lock set by

any other process or ODBC application. Also set the client **Timeout=** directive to zero (wait indefinitely) to ensure that it will not time out while waiting for a response from the U/SQL Server.

Accessing the UDD System Tables

Using the Interactive U/SQL utilities, [Win U/SQLi](#) or UNIX based [usqli](#), you can access the UDD system tables to obtain a better understanding of their contents. The table UDDTABLES contains details of all the UDD tables and you can obtain a list of them by performing the following query:

```
select tabname from uddtables;
```

This retrieves, from the Books Wholesaler demonstration example Single-tier **booksw32.udd** or Multiple-tier **books.udd** for a COBOL data source, the following:

```
tabname
-----
UDDTABLES
UDDCOLUMNS
UDDINDICES
UDDSOURCES
UDDVIEWS
UDDTABAUTH
UDDCOLAUTH
UDDUSER
UDDAUTH
ODBC_SPECIAL_COLUM
ODBC_TYPES
ODBC_STATISTICS
UFD_PATHNAMES
DICTIONARY_REVISIO
UDDLOG
GEN$$_TABLE
GEN$$_FILE
GEN$$_FIELD
GEN$$_INDEX
COBOL_TABLE
COBOL_FIELD
COBOL_INDEX
ARRAY_OBJECT
ARRAY_LEVEL
BUDGET
OLINE
ORDERS
STOCK
SALEHIST
CUSTOMER
```

```
30 records retrieved
```

All the tables following ARRAY_LEVEL are the application's tables. Differing data sources will have similar tables to **COBOL_TABLE**, **COBOL_FIELD** and **COBOL_INDEX**, for example C-ISAM has **CISAM_TABLE**, **CISAM_FIELD** and **CISAM_INDEX**.

Most UDDs have the same remaining system tables, except for the Transoft U/FOS database management system data source, where additional system tables are required. For further details see the section [Universal File Dictionary Overview](#).

In our 'Books' example, you use the following query to establish the application's data tables:

```
select tabname from cobol_table;
```

which outputs:

```
tabname
-----
STOCK
CUSTOMER
ORDERS
OLINE
SALEHIST
BUDGET
```

6 records retrieved

You use the following query to establish the columns in a particular data table:

```
select fldname from cobol_field where tabname="STOCK";
```

which outputs:

```
fldname
-----
STKNUM
STOCK_TITLE
PUBLISHER
AUTHOR_NAME
CATEGORY
PRICE
QTY
```

7 records retrieved

And you use the following query to establish the indexes for a particular data table:

```
select tabname, idxname, idxno, duplicates, fld1, fld2, fld3,
fld4 from cobol_index where tabname="STOCK"
```

This query outputs:

| TABNAME | IDXNAME | IDXNO | DUPLICATES | FLD1 | FLD2 | FLD3 | FLD4 |
|---------|---------|-------|------------|----------|------|------|------|
| STOCK | STOCKK0 | 0 | 101 | STKNUM | - | - | - |
| STOCK | STOCKK1 | 1 | 100 | CATEGORY | - | - | - |

2 records retrieved

Where:

- Index numbers, IDXNO, start from 0 (zero).
- If duplicates index entries are allowed the DUPLICATES entry value is '100', while for non-duplicates the value is '101'.
- There can be up to 64 key components in each index, that is FLD1 to FLD64.

Summary of UDD System Tables

You can review the contents of each UDD system table using the Interactive U/SQL utilities, [Win U/SQLi](#) or UNIX based [usqli](#), by issuing a query of the form:

```
select * from <udd_table_name>;
```

The following is a summary of the contents of each UDD system table.

- **UDDTABLES**

This table contains a list of all the tables contained in the UDD, with their ownership, table ID, table type, number of columns, number of indexes and number of rows (**nrows**).

You can affect how the [Query Planner](#) will operate by setting the value of **nrows**.

- **UDDCOLUMNS**

This table contains a list of column names for each table ID (defined in UDDTABLES), with the column number within the UDD table, the column type, its length, any decimal scale and whether or not it can contain 'null'.

- **UDDINDICES**

This table includes for each index name, its owner, the table ID to which it belongs, the index type, the number of component parts making the key and column IDs for each of up to 64 parts.

- **UDDSOURCES**

This table includes for each application data table its data source, for example Micro Focus COBOL (defined as MF) or C-ISAM (defined as C-ISAM).

- **UDDVIEWS**

This table is for future use.

- **UDDTABAUTH and UDDCOLAUTH**

These tables are not used.

- **UDDUSER and UDDAUTH**

These tables are optional for GRANT and REVOKE security. UDDUSER contains each username and password combination. UDDAUTH contains the access privileges of users to the application's tables and their columns. They can only be viewed by the dba, Database Administrator. See the [Security](#) section.

- **ODBC_SPECIAL_COLUM, ODBC_TYPES and ODBC_STATISTICS**

These table are for U/SQL internal use only.

- **UFD_PATHNAMES**

This table contains a list of directory paths, where the first row has a **pathid** of zero. This table is populated when a pathname is entered against a physical data file. However, for greater flexibility it is recommended that pathnames are not maintained in the UDD, but rather the **Searchlist** directive is used.

- **DICTIONARY_REVISIO**

This table contains the revision of the UDD. An error message is displayed if the UDD is 'too old' for the U/SQL Server.

If this is the case, then for non-COBOL data sources use your existing UFD text file to recreate your UDD, with the latest revision number, as described in the section [Creating the UDD from the UFD Text File](#).

For COBOL data sources you can export your existing UDD to a text file and then import it into a new UDD, with the latest revision number, as described in the section [Exporting and Importing your UDD](#).

- **GEN\$\$_TABLE, GEN\$\$_FILE, GEN\$\$_FIELD and GEN\$\$_INDEX**

These special UDD system tables cannot be accessed.

- **UDDLOG**

This table contains the definition of the log file, **usqlcs.log**.

- **XXXXX_TABLE, XXXXX_FIELD and XXXXX_INDEX**

These UFD tables have different names for 'XXXX' dependent on the data source, for example, COBOL, CISAM, UFOS and BB.

XXXXX_TABLE contains each application table name and the corresponding physical data file.

XXXXX_FIELD contains details of each column in each application table, although there will be differing information dependent on the data source.

XXXXX_INDEX contains details of each application table's indexes.

Refer to the details in the above section Accessing UDD System Tables.

- **ARRAY_OBJECT and ARRAY_LEVEL**

These tables are only available for certain data sources and contain repeating group information, for example, COBOL OCCURS.

Hints and Tips

This section contains various miscellaneous hints and tips.

Improve UDD Performance

The directive, **CacheNumPages**, determines the size of the buffer pool cache (in pages) that manages the UDD. For Single-tier U/SQL the default is 256 pages and for Multiple-tier the default is 64 pages.

If you have a large UDD there may be benefit in increasing the value of **CacheNumPages**. This reduces the number of times the U/SQL Server needs to access the UDD from disk. This is likely to be a benefit particularly for Single-tier installations.

However, with Multiple-tier installations you need to be careful since any increase in the size of the **CacheNumPages** cache will be required by each child process. This could also conflict with the more efficient operating system paging mechanism.

SQLDriverConnect()

If you are creating applications that require GRANT and REVOKE usernames and passwords to be processed, then this can be achieved by the ODBC function `SQLDriverConnect()` which has been extended. Refer to the [SQLDriverConnect\(\)](#) section.

Create Tables in UDD

Although it is not possible to use the **CREATE TABLE** statement to create new underlying data source tables, it is possible to create tables inside the UDD itself.

There are two type of table that can be created in the UDD:

- Normal tables using the **CREATE TABLE** statement.
- Temporary tables using the special **CREATE TEMP TABLE** statement. These tables only exist for the length of the session, that is until the connection with the UDD is dropped. It is also possible to re-issue a **CREATE TEMP TABLE** statement for the same table name without first issuing a **DROP TABLE** statement.

You can issue **CREATE INDEX** statements on both types of UDD table to improve the subsequent access performance of the tables.

These tables can be useful as sometimes it is impossible to obtain a query directly from the underlying application data.

The following is an example of both types of table created in the UDD and can be invoked via the Interactive U/SQL utilities, [Win U/SQLi](#) or the UNIX, [usqli](#):

```
# create example tables and populate test data
# -----
create table emp (eno integer, name char (20));
create table empadd (eno integer, addr char (20));
create table emptel (eno integer, telno char (10));
insert into emp values (1,'Jim Smith);
```

```

insert into emp values (2,'Fred Bloggs');
insert into emp values (3,'Jimmy Hall');
insert into emp values (4,'John Cannon');
insert into emp values (5,'Fred Jones');
insert into empadd values (1,'1 High St');
insert into empadd values (3,'2 Any Road');
insert into empadd values (5,'The Big House);
insert into emptel values (1,'123 4562');
insert into emptel values (2,'456 7891');
# create temporary (only this session) tables
# and insert each outer join (EMP X EMPADD) and (EMP # X EMPTEL) into
them
# -----
create temp table temp1 (eno integer, addr char (20));
create temp table temp2 (eno integer, telno char (20));
insert into temp1 select emp.eno,addr from {oj emp left outer join
empadd on empadd.eno=emp.eno};
insert into temp2 select emp.eno,telno from {oj emp left outer join
emptel on emptel.eno=emp.eno};
# finally this is the query which is the join of EMP # with the 2
outer-joins
# -----
select emp.eno,name,addr,telno from emp,temp1,temp2
where temp1.eno=emp.eno and temp2.eno=emp.eno;

```

Selection of Numeric columns defined as type CHAR

When a WHERE clause contains a COLUMN=<numeric> component and the COLUMN is of type CHAR, the numeric is converted to alpha, right justified and space filled to the column width.

If a numeric type field is re-defined over an alpha field and the alpha field is in the key then a special \$MAP entry in the record expression for a table will 'map' the numeric field to the alpha field in the WHERE clause. For example, assume:

```
rectype = 2 and $MAP (KEYN, KEYA)
```

in the record expression then any reference to KEYN=123 will be silently translated into KEYA=123. This solves the problem of Microsoft Query 'wrapping' the numeric value in quotes.

Division by Zero

Normally an error is returned when division by zero occurs. However, it is possible to specify a return value of either zero or NULL. This is achieved via a directive **DivZeroRtn** in either the [**Data Source Defaults**] section for Multi-tier, [**Transsoft U/SQL Defaults**] section for Single-tier and/or in a specific [**<data_source_name>**] section, with the specific entry taking precedence.

The valid entries are:

DivZeroRtn=0 - for a return of zero

Transoft U/SQL User Guide

DivZeroRtn=1 - for a return of a NULL

DivZeroRtn=9 - error returned

Troubleshooting

Troubleshooting

This section describes error messages that can be displayed and checks and tests that you can undertake to attempt to remedy problems.

A facility has been added to the U/SQL Client based Interactive U/SQL utility, [Win U/SQLi](#), that allows you to easily query the log file, **usqlcs.log** in which you can determine the level of messages and the details required.

It is recommended you read the Release Notes from the U/SQL Client program group on your PC and, for Multiple-tier UNIX versions, the **README** file in the base directory of your U/SQL Server software installation directory, by default **/usr/usqls**. Use either:

`more README` : to display it, or

`lp README` : to print it

For Multiple-tier Windows NT Server versions, read the U/SQL Server Release Notes in the U/SQL Adapters program group.

These Release Notes and README file may provide further information and assistance not provided with this Help file.

Note: *If you are having problems using the U/SQL Manager, for COBOL data sources only, then refer to the [U/SQL Manager Error Messages](#) section.*

If you have problems with your U/SQL Client or Server licensing, then refer to the [Licensing](#) section.

Introduction

If you are having problems:

1. Check your PC system requirements: see Chapter 3, How to Install the Single or Multiple-tier U/SQL Client in the Installation & Licensing guide for information.
2. Ensure you have set up the U/SQL Client directives and, for Multiple-tier, the U/SQL Server directives: see Chapters 2 and 3 of this manual for Single-tier and Multiple-tier respectively.
3. Ensure that U/SQL Server is running (Multiple-tier versions): see Chapter 4 of this manual.
4. The next section 32-bit U/SQL Administrator Test & Review Facilities provides useful assistance.

You can test connect to a UDD to ensure you have setup the directives correctly, before attempting to use an ODBC-enabled product. If you fail to connect details of the connection failure are displayed. You can view the ODBC drivers installed to ensure you have the correct one(s) installed. You can invoke the System & File Information utility, that provides information to check that your U/SQL Client is installed successfully.

5. Refer to the following parts in this section, according to the problem you have:
 - Problems Getting Started with an ODBC-enabled Product

Transoft U/SQL User Guide

- COBOL and General Issues
- PC Configuration, TCP/IP Connectivity and Your Network
- Error Messages, the Log File and How to Query it
- SQL Error Messages.

Appendices

Appendix A - Support Information

For support and information on this and other Transoft products and services, contact Transoft:

Transoft Atlanta, GA

1165 Northchase,
Suite 375
Marietta
GA 30067
USA

Tel: +1 (770) 933 1965
Fax: +1 (770) 933 3464

Transoft Dayton, OH

7333 Paragon
Suite 250
Dayton
OH 45459
USA

Tel: +1 (937) 438 5553
Fax: +1 (937) 438 5377

Transoft UK

Transoft House
5J Langley Business Centre
Station Road
Langley
Slough
SL3 8DS
England

Tel: +44 (0) 1753 778880
Fax: +44 (0) 1753 773050

Please use email to contact us wherever possible:

Europe, Middle East and Africa: uksupport@transoft.com

America and Rest of the World: ussupport@transoft.com

Web site: www.transoft.com

Appendix B - Sample License Form

To: **Joe Smith**
 Company: **ABC Company**
 Our Reference: **UK01350**

The following U/SQL Client-Server License is issued under a License Agreement:

To Organization: **ABC Company**
 For Features: **Multi-Tier, Read-Write**
 Data Source Driver(s): **Micro Focus COBOL**
 ODBC-products allowed: **Any**
 Server platform: **HP-UX**

Normally the following license details will be entered during installation of the U/SQL Client-Server software. However, they can be entered at any time.

Please refer to the Installation & Licensing Guide.

4 U/SQL Server Connections license. On your server, please use the U/SQL Server [utility 'serv_setup.sh', on UNIX, and 'U/SQL Service Manager', on Windows NT Server, to enter:]

Your Organization: **ABC Company**
 Server License Code: **S43A-U4-D5-F2-C1FGRT2TS**

4 U/SQL Client ODBC Driver (including Administrator) copy license

Please use the U/SQL Client License Tool to install a single License Code on a maximum of 4 individual client PC(s). From the License Tool, click the License Form button, enter Your Name followed by Your Organization, which is:

ABC Company

followed by **one** of the License Codes below: (remember to register each user's name)

| | Your License Codes | Register each user's name |
|----------|------------------------|---------------------------|
| User #1: | S43A-U4-F2-1-C2DFERTGN | |
| User #2: | S43A-U4-F2-2-CNBUYTDRT | |
| User #3: | S43A-U4-F2-3-CNJU6Y7HJ | |
| User #4: | S43A-U4-F2-4-C6GPLDRNU | |

2 U/SQL Manager copy license (each Upgrading an existing license)

Please use the U/SQL Client License Tool to upgrade the existing License Code on a maximum of 2 individual client PC(s). From the License Tool, click the Upgrade button, enter one of the License Codes below: (remember to register each user's name)

| | Your License Codes | Register each user's name |
|-------------|-------------------------|---------------------------|
| Upgrade #1: | S43A-UG1-F3-1-CJ7HYWSDI | |
| Upgrade #2: | S43A-UG1-F3-2-CJ23NHWK2 | |

Appendix C - U/FOS data type enhancements

The following data types have been introduced into the U/FOS Data Source Driver:

- **DG Julian** - The Julian Year 2000 date format is the Data General variant based on the number of days since 31 Dec 1967 in a 4 byte and 2 byte computational field. This date is the output of the COB-FDAY call in the AIM extension library and gives valid results between,1 Jan 1968 and 31 Dec 2145
- **YYMMDD** - The six digit date format. The **FixedDateOffset=nn** directive is supported. This allows you to define a 'cut-off' year below which dates with two digit years are considered to be in the 21st century. The directive can be set in either the data source default section of the **usqlsd.ini** configuration file for UNIX platforms or in the **USQLSD.INI Registry** folders for Windows NT Server
- **CCYYMMDD** - The fully compliant Year 2000 date format.

To make use of this new functionality, you must modify the **ufos_field** section of the U/FOS UFD file as illustrated below:

| objname | fldno | fldname | picture | fusage |
|----------------|--------------|----------------|----------------|---------------|
| UF85IX | 1 | FLD1 | YYMMDD | DATE(S6) |
| UF85REC | 1 | FLD1 | YYMMDD | DATE(S6) |
| UF85REC | 2 | FLD2 | YYMMDD | DATE(S6) |
| UF85REC | 3 | FLD3 | CYYMMDD | DATE(S8) |
| UF85REC | 4 | FLD4 | S9(9) | DATE(CDAY) |
| UF85REC | 5 | FLD5 | S9(4) | DATE(CDAY2) |
| UF85REC | 6 | FLD6 | X(15) | DISPLAY |

Appendix D - ACUCobol Configurable Variables

Following is a current list of configurable variables that may be set:

CARRIAGE_CONTROL_FILTER tinyint
COMPRESS_FACTOR tinyint
DUPLICATES_LOG string
EOL_CHAR tinyint
LOCKED_RECORD_DELAY† short
LOCKING_RETRIES† short
LOCKS_PER_FILE short
LOGGING bool
LOG_BUFFER_SIZE short
LOG_DEVICE bool
LOG_DIR string
LOG_ENCRYPTION bool
LOG_FILE string
MAX_FILES short
MAX_LOCKS short
NO_LOG_FILE_OK bool
OPEN_FILES_ONCE‡ bool
REL_DELETED_VALUE short
TRX HOLDS LOCKS bool
USE_LARGE_FILE_API‡ bool
V_BASENAME_TRANSLATION bool
V_BUFFERS long
V_BUFFER_DATA bool
V_BULK_MEMORY long
V_CACHE_FLUSH_PERCENT short
V_FORCE_OPEN bool
V_INDEX_BLOCK_PERCENT short
V_INTERNAL_LOCKS bool
V_LOCK_METHOD short
V_MARK_READ_CORRUPT bool
V_NO_ASYNC_CACHE_DATA bool
V_READ_AHEAD bool
V_SEG_SIZE long
V_STRIP_DOT_EXTENSION bool
V_VERSION short
WAIT_FOR_ALL_PIPES bool

Appendix E - MFOCUS Configurable Variables

No U/SQL directives are required to set this. Following example shows how configurable options may be set.

Set system variable EXTFH to contain a full path of the MF file handler configuration file.

```
EXTFH = C:\windows\system32\myextfh.cfg
```

The File Handler configuration file enables you to alter File Handler parameters for individual files, or for all files. Settings which apply to all files are listed under the tag:

```
[XFH-DEFAULT]
```

while settings which apply to an individual file are listed under the individual filename, for example:

```
[TEST.DAT]
```

The settings for an individual file override the global settings.

Note: If you have duplicate name=value pairs under a [tag] in your File Handler configuration file, then only the first entry is used.


For more information on Configuration options see:

<http://supportline.microfocus.com/documentation/books/sx20books/fhcnfg.htm>

Printed Documentation

To view a printable pdf version of this online Help file, click the following button:

This will launch the **Adobe Acrobat Reader**, which will display the pdf version of this online Help file. To print the file either:

- Select the **Print** command from the **File** menu
- Click the Print icon  on the toolbar.

Note: *If you downloaded the **USQL_Help.chm** online Help file from the Transoft Website, you need to also download the **USQL_User_Guide.pdf** file into the same directory, to benefit from the functionality described above.*

Index

| | |
|--|---|
| [| CachePageSize (ODBC.INI Directive) |
| [<Data_source_name>] Section | 148 |
| [Transoft U/SQL Configuration] | |
| Section..... | 140 |
| [Transoft U/SQL Defaults] Section | |
| | 143 |
| 2 | |
| 2-user Temporary License | 41, 52 |
| A | |
| Access Methods Supported | 257 |
| Accessing the UDD System Tables | |
| | 441 |
| ACUCOBOL Vision 4 data files | 353 |
| Advanced Directives..... | 150 |
| Advanced Use | |
| U/SQL Adapters..... | 437 |
| Advanced Use | 437 |
| Advanced Use of U/SQL Adapters | 437 |
| AlternativeIndex (Directive)..... | 203 |
| Amending | |
| Data Dictionary | 351 |
| Amending | 351 |
| Amending the Data Dictionary... 229, | 351 |
| Array Elements | |
| Mapping | 237 |
| Array Elements | 237, 240, 344 |
| Array Handling | 411 |
| B | |
| BBASIC ISAM Data Dictionary | |
| Defining | 376 |
| BBASIC ISAM Data Dictionary | 376 |
| BBLockType (Directive)..... | 395 |
| BecomeUser (Security Directive) | 187 |
| BETWEEN Predicate | 233, 260 |
| Business BASIC Data Dictionary . | 375 |
| C | |
| CacheNumPages (ODBC.INI | |
| Directive) | 140 |
| CacheTables (Directive)..... | 203 |
| C-ISAM Data Dictionary | 357 |
| C-ISAM Data Types | 364 |
| Client License | 36, 52, 277 |
| Client ODBC.INI Directives | 278 |
| Client Software | |
| Installing | 12 |
| Client Software..... | 4, 10, 12, 36, 52, 108, 109, 126, 149, 171, 215, 229, 278, 349, 439 |
| COBOL .. 1, 4, 10, 12, 36, 41, 42, 79, | 81, 86, 94, 98, 106, 107, 108, 119, 121, 126, 137, 140, 143, 150, 161, 166, 182, 185, 192, 203, 215, 229, 236, 237, 240, 253, 255, 258, 262, 265, 267, 268, 273, 274, 276, 280, 285, 288, 290, 292, 310, 312, 314, 317, 318, 326, 333, 338, 339, 344, 349, 353, 398, 400, 408, 411, 439, 441, 448 |
| COBOL FD Converter Error Messages | |
| | 333 |
| COBOL FD File | |
| Selecting..... | 285 |
| COBOL FD File | 285 |
| COBOL Level | 140, 292 |
| COBOL UDD | |
| Manually Creating | 338 |
| COBOL UDD | 338 |
| COGNOS Impromptu Outer Join | |
| Directives..... | 160 |
| COGNOS Impromptu support for | |
| outer joins | 150 |
| Comma Delimited UFD File..... | 339 |
| Comparison Predicate... 67, 233, 260 | |
| Configuring the EJE on UNIX | 425 |
| createhjeudd (Directive)..... | 418 |
| Creating | |
| New Data Dictionary..... | 280 |

| | | | |
|--|---|---|--|
| UDD..... | 229 | Description (ODBC.INI Directive) | 148 |
| Creating | 229, 280 | Dictionary (ODBC.INI Directive).. | 143 |
| Creating a Dictionary | 280 | Distinct Jump (Directive) | 203 |
| Creating a UDD ... | 80, 229, 253, 268, 273, 277, 280, 333, 357, 375, 396 | Driver (ODBC.INI Directive) | 148 |
| Creating the UDD from the UFD Text File | 229, 386 | DSDrivers (INI Directive)..... | 148 |
| D | | E | |
| Data Arrays | | EJE | 416, 418, 425, 427, 431, 435 |
| Handling..... | 236 | Entering the ODBC.INI Directives.. | 81 |
| Data Arrays | 236, 237, 246 | Enterprise Join Engine | |
| Data Dictionary | | Using..... | 418 |
| Amending | 351 | Enterprise Join Engine | 416, 418 |
| Data Dictionary ... | 1, 10, 74, 81, 121, 143, 187, 192, 215, 227, 229, 253, 255, 273, 274, 280, 292, 317, 322, 330, 333, 338, 339, 351, 357, 358, 364, 376, 397, 408, 418, 425, 439 | Error Message File (usqlcs.msg) . | 166 |
| Data Objects..... | 398, 406, 408 | Error Messages..... | 4, 12, 36, 47, 52, 126, 140, 143, 166, 192, 441 |
| Data View..... | 310 | Error Reporting Format..... | 166 |
| Database Administrator..... | 186, 192, 441 | Example Data Dictionary..... | 408 |
| Dealing with Multiple 01 Level Records | 258 | Existing Table | |
| Define Fields (Columns) | 344 | Delete | 325 |
| Defining | | Existing Table..... | 324, 325, 326 |
| BBASIC ISAM Data Dictionary . | 376 | Exporting and Importing your UDD | 126, 330 |
| C-ISAM Data Dictionary..... | 358 | Expression Handling..... | 233 |
| COBOL Data Dictionary..... | 339 | Expression Handling Syntax | 260 |
| NULL Columns..... | 264 | Expression Syntax | 233 |
| Defining | 264, 339, 358, 376 | Expressions on Columns and Virtual Columns | 265 |
| Defining a BBASIC ISAM Data Dictionary..... | 376 | F | |
| Defining a C-ISAM Data Dictionary | 358 | FHPassword (ODBC.INI Directive) | 140 |
| Defining a COBOL Data Dictionary | 339 | FHREDIR (ODBC.INI Directive) ... | 140 |
| Defining Datatables | 341 | FHUsername (ODBC.INI Directive) | 140 |
| Defining NULL Columns (Fields).. | 264 | Field (Column) Names..... | 262 |
| Delete an Existing Table | 325 | File Data Sources..... | 74 |
| Demonstration application | 10, 12, 79, 86, 215 | File Details..... | 314 |
| Descending Keys | 358 | File DSN | 74 |
| | | File DSNs..... | 74 |
| | | FileNamePrefix (ODBC.INI Directive) | 143 |
| | | FileNameSuffix (ODBC.INI Directive) | 143 |
| | | Fileshare..... | 140, 150, 182, 353 |
| | | FixedDateOffset (ODBC.INI Directive)..... | 143 |

- Flattened Array 236
- Flattened Array237, 239, 240
- ForceSimilarIndex (Directive)..... 203
- Foreign Character Set Support 98, 143, 149
- G**
- GRANT and REVOKE Security 126, 166, 192, 227, 441
- Grant/Revoke security 192
- H**
- Handling of Data Arrays 236
- HideSystemTables (ODBC.INI Directive) 143
- I**
- Indexes67, 116, 119, 126, 140, 150, 170, 171, 192, 198, 203, 213, 229, 236, 237, 240, 253, 255, 257, 268, 285, 288, 292, 312, 314, 317, 318, 321, 330, 338, 339, 341, 344, 348, 349, 353, 357, 358, 374, 376, 386, 398, 400, 407, 408, 416, 439, 441
- IndexStats (Directive)..... 203
- Installation and Licensing U/SQL 3
- Installing
 - Client Software..... 12
 - Multiple-Tier U/SQL Server 4
 - Server License..... 42
 - Single or Multiple-tier U/SQL client 10
 - U/SQL Server License..... 41
- Installing.....4, 10, 12, 41, 42
- Installing the Client Software 12
- Installing the Multiple-tier U/SQL server 4, 10
- Installing the Server License on UNIX Platforms..... 42
- Installing the Server License on Windows NT Server..... 47
- Installing the Single or Multiple-tier U/SQL client..... 10, 448
- Installing the U/SQL Server License4, 41, 52
- Interactive U/SQL Utilities 10, 41, 79, 108, 109, 126, 143, 161, 166, 192, 229, 330, 349, 386, 431, 448
- Issues 4, 41, 106, 121, 143, 182, 186, 192, 236, 292, 353, 386, 416, 435, 439, 441, 445, 448
- Issues and Limits..... 439
- J**
- JDBC Client..... 136
- JDBC Driver Support 136
- K**
- Keys Definition 312, 314
- L**
- License Form (sample) 452
- Licensing 35
- LIKE Predicate..... 67, 233, 260
- Limits.. 52, 110, 113, 121, 187, 198, 240, 251, 292, 312, 333, 357, 416, 439
- Line Sequential.....257, 314, 341
- Linking External File Handlers..... 107
- Lockdown security 187
- Locking..... 185
- Locking under U/BL..... 395
- Log File Messages 161
- Log Levels ...30, 121, 140, 161, 166, 278
- LogFileDir 86, 102, 140, 161
- LogFileDir (ODBC.INI Directive).. 140
- Logical Operators.....233, 260
- Logival Operators..... 260
- LogLevel.....86, 102, 140, 149, 158, 161, 166
- LogLevel (ODBC.INI Directive) ... 140
- M**
- Manipulate
 - Data..... 290
- Manipulate...63, 170, 171, 215, 229, 253, 263, 265, 290, 292, 339, 358, 386
- Manually Creating a COBOL UDD265, 338
- Mapping
 - Array Elements 237
- Mapping 237
- Mapping Array Elements to a Relational Model 237

| | | |
|-----------------------------------|---|-------------------------------------|
| Micro Focus COBOL Fileshare | 353 | 108, 116, 121, 126, 137, 139, |
| Micro Focus COBOL Specific Issues | | 140, 149, 150, 160, 166, 170, |
| | 106 | 171, 182, 192, 215, 222, 233, |
| Modifying | | 251, 263, 278, 280, 292, 333, |
| UDD..... | 321 | 394, 416, 425, 427, 431, 439, |
| Modifying | 321 | 445, 448 |
| Modifying a UDD . | 81, 143, 292, 310, 321 | ODBC Access to Non-relational Files |
| MsgFileDir | 86, 102, 140 | |
| MsgFileDir (ODBC.INI Directive) . | 140 | 63 |
| Multi-Byte NULL Expressions for | | ODBC API conformance |
| COBOL data types..... | 353 | 67 |
| Multi-company Support | 438 | ODBC Compliance..... |
| Multiple Record Types Under the | | 67 |
| Same Sub-index..... | 415 | ODBC Components..... |
| Multiple-tier model of U/SQL.... | 3, 63 | 63 |
| Multiple-tier Security... | 126, 187, 192 | ODBC Error Handling..... |
| Multiple-tier Security - User | | 166 |
| Connection | 187 | ODBC Error Messages..... |
| N | | 333 |
| Nested OCCURS . | 237, 246, 253, 268, 292 | ODBC Overview |
| New Data Dictionary | | 63 |
| Create..... | 280 | ODBC.INI Directives |
| New Data Dictionary | 74, 121, 253, 280 | Entering..... |
| NewDictionaryDir..... | 86, 102, 140 | 81 |
| NewDictionaryDir (ODBC.INI | | ODBC.INI Directives..... |
| Directive) | 140 | 79, 81, 108, |
| Newline..... | 257 | 137, 333 |
| NULL Columns | | ODBCVer |
| Defining | 264 | 140 |
| NULL Columns..... | 264 | ODBCVer (ODBC.INI Directive)... |
| Num2Char (ODBC.INI Directive). | 140 | 140 |
| O | | ODOSLIDE .. |
| OCCURS.... | 237, 239, 240, 246, 253, 258, 268, 280, 288, 292, 312, 314, 344, 411, 441 | 268, 280, 292, 314, 341 |
| OCCURS depending | 237 | Open an Existing Data Dictionary |
| OCCURS Depending on..... | 246, 268 | 322 |
| OCCURS Flattened Out into | | Open DataBase Connectivity |
| Individual Unique Fields..... | 268 | 63 |
| ODBC.... | 1, 3, 10, 12, 36, 57, 63, 67, 73, 74, 79, 80, 81, 94, 98, 101, | OpenExclusive (ODBC.INI Directive) |
| | | |
| | | 143 |
| | | P |
| | | Parallel OCCURS |
| | | 237, 246, 253, 268, |
| | | 292 |
| | | PartialIndex (Directive)..... |
| | | 203 |
| | | Pattern Syntax |
| | | 233, 260 |
| | | PICTURE..... |
| | | 292, 333 |
| | | Q |
| | | Query |
| | | Loading |
| | | 118 |
| | | Saving..... |
| | | 118 |
| | | Query..... |
| | | 1, 10, 24, 30, 67, 79, 108, |
| | | 109, 110, 113, 116, 118, 119, |
| | | 121, 126, 143, 149, 150, 161, |
| | | 170, 171, 181, 182, 185, 186, |
| | | 192, 198, 203, 213, 215, 222, |
| | | 229, 233, 239, 240, 246, 255, |
| | | 260, 264, 280, 292, 344, 349, |
| | | 374, 376, 386, 416, 418, 435, |
| | | 441, 445, 448 |
| | | Query Plan |
| | | Affecting |
| | | 198 |

- selecting..... 198
- Viewing 198
- Query Plan. 110, 113, 126, 150, 198, 203, 213
- Query Planner
 - Tuning..... 203
- Query Planner..... 1, 198, 203, 213
- Query Planner Hinting 213
- Querying and Manipulating Data. 171
- Querying the Log File. 109, 110, 113, 161, 229, 448
- R**
- READ_ONLY Views..... 179
- ReadOnly (ODBC.INI Directive) .. 143
- READ-ONLY Views 170
- Record Number
 - Retrieving..... 394
- Record Number ..333, 344, 374, 376, 394
- Record Number Access
 - Support..... 374
- Record Number Access..... 374
- REDEFINES. 253, 255, 267, 288, 292
- Relational View
 - U/FOS Data 397
- Relational View 1, 227, 397
- Relational View of U/FOS Data ... 397
- Rename an Existing Table..... 324
- Rename Table Names 287
- Repeating Group .236, 246, 268, 441
- Repeating Groups.....233, 237, 240, 268, 292
- Representing COBOL Data . 253, 255, 274
- Retrieving
 - Record Number (R1) using SQLGet 394
- Retrieving..... 394
- Return/newline..... 257
- Review
 - UFD Tables 400
- Review...36, 79, 108, 288, 353, 400, 441, 448
- S**
- Same record type Under Multiple Subindexes 414
- Sample License Form36, 452
- Sample Queries 171
- SearchList (ODBC.INI Directive) . 143
- Security.....24, 30, 86, 92, 102, 117, 126, 140, 158, 186, 187, 192
- Security (ODBC.INI Directive) 150
- Security (Security Directive) 187
- Security Directive 158
- Security directives 150
- SecurityFile (ODBC.INI Directive) 150
- SecurityFile (Security Directive).. 187
- SecurityFile Directive..... 158
- Selecting
 - COBOL FD File 285
- Selecting 285
- Server License
 - Installing on Windows NT.....47
- Server License.....42, 47, 52, 101
- Service Manager20, 203
- SetInIndexUse (Directive)..... 203
- Setting Up the UNIX usqlsd.ini File 102
- Single-tier installation files79
- Single-tier model of U/SQL..... 3, 63
- SQL conformance.....67
- SQL Error Messages 166
- SQL grammar coverage67
- SQL Syntax Supported 170
- SQLGetStmtOption.....67, 394
- start_serv.sh script, UNIX 101
- Starting and Stopping the UNIX Server 101
- Starting the U/SQL Manager 253, 278
- stop_serv.sh script, UNIX 101
- StripUnprintable (ODBC.INI Directive)..... 143
- Structure View290, 292
- Substitution (ODBC.INI Directive)143
- Support

| | | | |
|---|---|---|---------|
| ACUCOBOL Vision | 353 | Relational View | 397 |
| Support1, 4, 10, 61, 63, 67, 74, 109, 116, 117, 119, 126, 136, 140, 149, 150, 160, 170, 171, 179, 182, 192, 203, 222, 227, 229, 233, 236, 239, 240, 246, 251, 257, 260, 268, 280, 292, 310, 333, 353, 357, 358, 364, 374, 375, 376, 386, 406, 408, 416, 425, 451, 453 | | U/FOS Data 396, 397, 400, 406, 414, 453 | |
| Support for Record Number Access | 374 | U/FOS Data Dictionary | 396 |
| Support Information | 451 | U/FOS data type enhancements.. | 453 |
| Sustem Data Source | 73 | U/SQL1, 3, 4, 10, 12, 20, 24, 30, 35, 36, 41, 42, 47, 52, 57, 63, 67, 74, 79, 80, 81, 86, 92, 94, 98, 101, 102, 106, 107, 108, 109, 119, 121, 126, 136, 137, 140, 143, 149, 150, 161, 166, 170, 171, 179, 182, 186, 187, 192, 198, 203, 213, 215, 222, 227, 229, 233, 236, 240, 251, 253, 255, 257, 258, 260, 262, 263, 264, 268, 273, 274, 277, 278, 280, 288, 292, 312, 314, 319, 321, 322, 330, 333, 338, 344, 353, 357, 358, 364, 374, 375, 376, 386, 400, 406, 414, 416, 418, 425, 427, 431, 435, 437, 439, 441, 445, 448, 455 | |
| System and User Data Sources | 73 | U/SQL Adapters | |
| T | | Advanced Use..... | 437 |
| Table and column names error messages | 333 | U/SQL Adapters..... | 437 |
| TempDir (ODBC.INI Directive).... | 143 | U/SQL Administrator ... 1, 10, 12, 20, 36, 74, 80, 94, 108, 149, 203, 229, 425, 431, 448 | |
| TempTablePages (ODBC.INI Directive) | 140 | U/SQL Administrator Facilities | 94 |
| The Query Planner..... | 110, 113, 126, 198, 203, 213, 344, 358 | U/SQL Client Installation 12, 57, 108, 215 | |
| Transaction Processing | 182 | U/SQL Client Installation Files | 108 |
| Transaction Processing Syntax ... | 182 | U/SQL Client License | |
| TranslationDLL (ODBC.INI Directive) | 143 | Installing | 36 |
| TranslationFile (ODBC.INI Directive) | 143 | Removing | 36 |
| TranslationName (ODBC.INI Directive) | 143 | U/SQL Client License. 10, 12, 35, 36, 52 | |
| TranslationOption (ODBC.INI Directive) | 143 | U/SQL Enterprise Manager | 418 |
| Transoft 1, 4, 12, 52, 63, 67, 81, 94, 126, 136, 137, 160, 161, 166, 203, 213, 278, 394, 416, 418, 425, 427, 431, 441, 445, 451, 457 | | U/SQL Error Messages..... | 166 |
| Troubleshooting | 448 | U/SQL Manager ... 10, 12, 20, 36, 74, 79, 81, 108, 143, 192, 215, 251, 253, 255, 268, 273, 278, 280, 292, 312, 314, 319, 321, 322, 333, 338, 344, 353, 439, 448 | |
| TrueSFU (Directive) | 185 | U/SQL Manager Error Messages.. | 333 |
| TrueSFU (ODBC.INI Directive).... | 143 | U/SQL Record Mapper | 386 |
| Tuning | | U/SQL Server Directives | 92, 314 |
| Query Planner | 203 | U/SQL Server Installation on UNIX | 98 |
| Tuning | 203 | | |
| Tuning the Query Planner | 203 | | |
| U | | | |
| U/FOS Data | | | |

- U/SQL Server License
 - Installing 41
 - U/SQL Server License . 4, 35, 36, 41, 42, 47, 52, 86
 - U/SQL Server License Code ...36, 41, 42, 47
 - U/SQL Service Manager4, 24, 47, 84, 86, 92, 187, 277, 431, 435
 - UDD
 - Creating 229
 - Modifying..... 321
 - UDD ... 1, 10, 24, 30, 67, 74, 79, 81, 92, 94, 98, 101, 102, 108, 109, 110, 113, 119, 121, 126, 137, 139, 140, 143, 150, 170, 192, 198, 203, 215, 227, 229, 236, 237, 239, 240, 246, 251, 253, 255, 260, 273, 276, 280, 285, 292, 317, 319, 321, 322, 330, 333, 338, 341, 344, 351, 357, 364, 375, 376, 386, 396, 397, 400, 416, 418, 425, 439, 441, 445, 448
 - UDD Error Messages 333
 - UDD Overview..... 227
 - UDD System Tables
 - Accessing 441
 - UDD System Tables 441
 - UFD Tables 229, 265, 339, 341, 344, 348, 358, 376, 400, 408, 411
 - ufd_pathnames 400, 408
 - ufos_array_level..240, 246, 400, 411
 - ufos_array_object 240, 246, 398, 400, 411
 - ufos_field .. 239, 240, 246, 400, 408, 411, 453
 - ufos_index.. 398, 400, 408, 411, 414
 - ufos_index_level400, 408, 411
 - ufos_object 240, 246, 398, 400, 408, 411
 - ufos_tab_object ..240, 400, 408, 411
 - ufos_table .. 246, 400, 408, 411, 415
 - UnauthorizedAccess (Security Directive) 187
 - Universal Business BASIC..... 375
 - Universal Business Language..... 150, 375
 - Universal Data Dictionary Overview 227
 - Universal File Dictionary Overview 398
 - UNIX Client-Side Directives..... 139
 - UNIX Server
 - Starting 101
 - Stopping 101
 - UNIX Server..... 101, 126, 192, 314, 330, 349, 431
 - UNIX usqlsd.ini.....42, 52, 102, 182, 187, 386
 - UNIX usqlsd.ini File
 - Setting up..... 102
 - UNIX usqlsd.ini File52, 102
 - Upgrading UDDs 119
 - User Administration..... 192
 - User Data Source..... 73, 74, 137
 - Using Query Hinting 213
 - Using the Enterprise Join Engine. 418
 - USQLCS.LOG File 140, 161
 - usqlcs.msg.....98, 166
 - USQLCS.MSG file 140
 - usqli.... 98, 109, 126, 139, 187, 192, 198, 229, 330, 349, 353, 386, 425, 427, 431
 - usqli on UNIX Servers 126
 - usqli utility..... 126
 - usqlsd daemon 101
 - usqlsd.ini file..... 101, 102, 140, 143, 148, 149, 158, 161, 187, 251, 353, 425, 427, 431
- v**
- Validation of Security directives.. 150
 - Validation Rules..... 158
 - Validation Rules for Security Directives..... 158
 - View or Modify an Existing Table .. 326
 - Virtual Columns. 265, 292, 326, 344, 358

W

Win U/SQLi.....10, 12, 74, 109, 110,
113, 116, 117, 119, 121, 161,
192, 198, 203, 229, 349
Win U/SQLi Scripting..... 121
Win U/SQLi SQL Syntax..... 116

Win U/SQLi User Login Security .. 117
Windows U/SQL Server..... 24, 86
Writing Applications Using U/SQL
Client-Server To Access Your Data
..... 222

