

**Reality**  
V10.0

---

**Differences Supplement**

All trademarks including but not limited to brand names, logos and product names referred to in this document are trademarks or registered trademarks of Northgate Information Solutions UK Limited (Northgate) or where appropriate a third party.

This document is protected by laws in England and other countries. Unauthorised use, transmission, reproduction, distribution or storage in any form or by any means in whole or in part is prohibited unless expressly authorised in writing by Northgate. In the event of any such violations or attempted violations of this notice, Northgate reserves all rights it has in contract and in law, including without limitation, the right to terminate the contract without notice.

© Copyright Northgate Information Solutions UK Limited, 2003.

Document No. UM70006928AR1  
September 2003

Northgate Information Solutions UK Limited  
Peoplebuilding 2  
Peoplebuilding Estate  
Maylands Avenue  
Hemel Hempstead  
Herts  
HP2 4NW  
  
Tel: +44 (0)1442 232424  
Fax: +44 (0)1442 256454  
  
[www.northgate-is.com](http://www.northgate-is.com)

---

# Contents

## Chapter 1 About this Manual

Purpose of this Manual.....	1-2
Contents .....	1-2
Related Documents .....	1-2
Conventions.....	1-3
User Comments.....	1-5

## Chapter 2 New Features in Reality V10.0

GUI Administration .....	2-2
Foreign Database Support .....	2-3
MultiValue Compatibility .....	2-4
File Triggers.....	2-4
TCP Connections in DataBasic .....	2-5
Pseudo Floppy.....	2-5
Additional ACCOUNT-RESTORE Options .....	2-5
SYSTEM Statement.....	2-5
SP-ASSIGN .....	2-5
Rapid Recovery File System .....	2-7
Compressed Tape Image.....	2-8
Support for Distributed Transactions under MTS/COM+ .....	2-9
Other New Features .....	2-10
Large Databases .....	2-10
Networking.....	2-10
RealEdit .....	2-10

## Chapter 3 GUI Administration

GUI Administration .....	3-2
Tree Structure.....	3-3
Features.....	3-4
Logging In .....	3-5
Setting Up and Maintaining Databases .....	3-5
Backup and Restore .....	3-5

Users .....	3-6
Security Profiles .....	3-6

**Chapter 4 Foreign Database Support**

Summary of Reality File Types .....	4-2
LISTFILES .....	4-3
Foreign Database Files (Reality-specific Storage Format).....	4-6
File Definition Item .....	4-6
FDB-CLEAR .....	4-9
FDB-SET .....	4-10
FDB-SHOW .....	4-11
Saving and Restoring .....	4-11
SQL View Files .....	4-12
SQL-VIEW .....	4-12

**Chapter 5 MultiValue Compatibility**

File Triggers .....	5-2
How to Write a Trigger Routine .....	5-2
Debugging Triggers .....	5-4
How to Associate a Trigger with a File .....	5-5
Commands that might run Triggers .....	5-7
Examples .....	5-8
Triggers Dos and Don'ts .....	5-10
TCP Connections in DataBasic .....	5-12
Example Programs .....	5-12
Pseudo Floppy Support .....	5-22
SP-ASSIGN Enhancements .....	5-23
Additional ACCOUNT-RESTORE Options .....	5-24
TCL Commands .....	5-25
CREATE-TRIGGER .....	5-25
DELETE-TRIGGER .....	5-26
FDISCTOTAPE .....	5-26
LIST-TRIGGERS .....	5-28
TAPETOFDISC .....	5-30

DataBasic Statements and Functions .....	5-32
ACCEPT Statement.....	5-32
CONNECT Statement.....	5-37
Debugger Commands .....	5-42
@ .....	5-42
M.....	5-43
WF .....	5-43
WS.....	5-44

**Chapter 6 Rapid Recovery File System**

Description of Rapid Recovery .....	6-2
What is Rapid Recovery? .....	6-2
How Rapid Recovery Works.....	6-2
Configuring a Database for Rapid Recovery.....	6-4
Recovery Procedure.....	6-6
Shadow Databases.....	6-7
Actions Following Rapid Recovery .....	6-7

**Chapter 7 Compressed Tape Image**

Tape Images.....	7-2
Data Compression .....	7-2

**Chapter 8 Support for Distributed Transactions under MTS/COM+**

Distributed Transactions.....	8-2
MDTC Recovery Process.....	8-5
rxaserver Command .....	8-6

**Index**

**List of Figures**

Figure 6-1. Transaction Logging Configuration and Setup Menu .....	6-4
Figure 8-1. COM+ Distributed Transactions.....	8-3

---

# **Chapter 1**

## **About this Manual**

This chapter describes the different sections of this manual and any conventions used.

## Purpose of this Manual

This manual summarises the differences seen by users upgrading from Reality V9.1 to V10.0.

### Contents

**Chapter 1, About this Manual**, describes the different sections of the manual and any conventions used.

**Chapter 2, New Features in Reality V10.0**, summarises the features that have been added in Reality V10.0 and describes in detail those features that are not covered elsewhere in this manual.

**Chapter 3, Administration Tool**, describes the GUI Administration Tool. This allows many administrative tasks to be carried out through a simple-to-use graphical interface.

**Chapter 4, Foreign Database Support**, describes how Reality can access data held on SQL-based databases.

**Chapter 5, MultiValue Migration**, describes new features in Reality that have been added to improve compatibility with other MultiValue system.

**Chapter 6, Rapid Recovery File System**, describes an additional resilience option that logs all changes to a database's structure, so that it is possible to return a database to a usable state within minutes of restarting after a system failure.

**Chapter 7, Compressed Tape Image**, describes how you can specify a compression level for data saved in a tape image.

**Chapter 8, Support for Distributed Transactions under MTS/COM+**, describes how Reality V10.0 provides support for distributed transactions through the ODBC and XA interfaces. It also explains the recovery procedures required when using distributed transactions.

### Related Documents

Reality on-line documentation.

On-line help for Reality GUI Administration Tool.

On-line help for RealEdit.

## Conventions

The following conventions are used in this manual:

<b>Text</b>	Bold text shown in this typeface is used to indicate input which must be typed at the terminal.
Text	Text shown in this typeface is used to show text that is output to the screen.
<b>Bold text</b>	Bold text in syntax descriptions represents characters typed exactly as shown. For example  <b>WHO</b>
Text	Characters or words in italics indicate parameters which must be supplied by the user. For example in  <b>LIST</b> <i>file-name</i>  the parameter <i>file-name</i> is italicized to indicate that you must supply the name of the actual file defined on your system.  Italic text is also used for titles of documents referred to by this document.
{ }	Braces enclose options and optional parameters. For example in  <b>BLIST</b> { <b>DICT</b> } <i>file-name item-id</i> {(options)}  The word <b>DICT</b> can optionally be typed to specify the dictionary of the file.  <i>file-name</i> and <i>item-id</i> must be supplied.  One or more single-letter options can be included, as defined for the command; these must be preceded by an open parenthesis, can be given in any order, and are not separated by spaces. Any number of options can be used except where specified in text.
[ <i>param</i>   <i>param</i> ]	Parameters shown separated by vertical lines within square brackets in syntax descriptions indicate that at least one of these parameters must be selected. For instance,  [ <b>THEN</b> <i>statements</i>   <b>ELSE</b> <i>statements</i> ]  indicates that either a THEN clause or an ELSE clause must be included (or both).
...	In syntax descriptions, indicates that the parameters preceding can be repeated as many times as necessary.
SMALL CAPITALS	Small capitals are used for the names of keys such as RETURN.

CTRL+X	Two (or more) key names joined by a plus sign (+) indicate a combination of keys, where the first key(s) must be held down while the second (or last) is pressed. For example, CTRL+X indicates that the CTRL key must be held down while the X key is pressed.
Enter	To enter means to type text then press RETURN. For instance, 'Enter the WHO command' means type <b>WHO</b> , then press RETURN.  In general, the RETURN key (shown as ENTER or ↵ on some keyboards) must be used to complete all terminal input unless otherwise specified.
Press	Press single key or key combination, but do not press RETURN afterwards.
X'nn'	This denotes a hexadecimal value.

## User Comments

A Comment Sheet is included at the front of this manual. If you find any errors or have any suggestions for improvements in the manual please complete and return the form. If it has already been used then send your comments to the Technical Publications Manager at the address on the title page, or email [techpubs@northgate-is.com](mailto:techpubs@northgate-is.com).

---

## **Chapter 2**

# **New Features in Reality V10.0**

This chapter summarises the features that have been added in Reality V10.0.

## GUI Administration

GUI Administration is a tool providing a Graphical User Interface for easy routine administration of the system. Via a tree-structure the user is able to:

- Set up and maintain databases including:
  - ❑ Security profiles.
  - ❑ Users.
  - ❑ Backup & restore.
- Administer the system
  - ❑ Add/remove system users.
  - ❑ Server administration.
  - ❑ Thread management.
  - ❑ Set up and administer user access to databases through the GUI Administration tool.

The tasks replace and supplement most of those available to the system administrator at TCL.

The interface runs on a client and allows different systems to be accessed via a tree structure to enable administrative tasks to be performed on Reality databases and their environments. It consists of a two-pane screen with the left-hand pane containing the tree structure representing the systems and their Reality databases, and the right hand pane containing tabbed sub-panes whose contents reflect the selected tree node. Tooltips are displayed when hovering over most fields but if more information is needed then tab-specific help can be displayed below the panes.

The GUI Administration Tool is described in greater detail in Chapter 3.

## Foreign Database Support

This feature provides Reality with access to data held on SQL-based foreign databases (currently Oracle or SQL Server). There are two ways of doing this, implemented as new Reality file types:

- Reality-specific storage format. In this format, the foreign database is set up to emulate Reality files, thus allowing Reality applications to store their data in the foreign database. The Reality account is set up in a local database, with some or all of the files held on a remote foreign database. The *file definition item* contains details of the file location.

A new verb, *FDB-SET*, is used to change the way in which the CREATE-FILE verb operates. Following execution of *FDB-SET*; subsequent CREATE-FILE operations take place within the specified foreign database. Subsequent restore operations will restore files onto this foreign database. CREATE-FILE and the restore commands work in this way until *FDB-CLEAR* is executed. The verb *FDB-SHOW* lets you view the current foreign database setting.

- Data exchange storage format (*SQL-VIEW*). This provides Reality with a view of data held in the foreign database. The data remains in the foreign database format and therefore only limited access is possible from Reality.

Foreign database support requires a working ODBC installation, appropriate ODBC driver(s) and Data Source Definition(s) on the Reality system.

Foreign database support is described in greater detail in Chapter 4.

## MultiValue Compatibility

Reality V10.0 has been further enhanced to improve compatibility with other MultiValue systems. The new features are summarised here and described in greater detail in Chapter 5.

### File Triggers

This feature allows the user to specify a DataBasic subroutine that will run automatically before a file item is written or deleted. A file trigger can be set to run before or after an item is written and before or after an item is deleted.

- Triggers that run before file operations are mainly used to validate the attempted change to the database against user-defined constraints, or “business rules”, and allow the change only if the constraint is satisfied.
- Triggers that run after file operations are normally used to create audit trails and other transaction logs.
- All types of trigger can be used to create relationships between files, to ensure that whenever one file is updated, another related file is also updated.

The components of the file triggers feature are as follows:

- A new DataBasic function, *ACCESS*, can be called from within a trigger subroutine. It returns information relating to the trigger, the file with which it is associated and the item being written or deleted.
- New debugger commands *@\**, *WF*, *WF\** and *WS* and additional options to the *SET-OPTION command* make it possible to debug triggers. They also make it easier to debug DataBasic programs called from *PERFORM* statement and *Procs*.
- New TCL verbs: *CREATE-TRIGGER*, *DELETE-TRIGGER*, *LIST-TRIGGERS*. The first of these allows you to associate a trigger subroutine with a Reality file and to specify whether it will run when an item is written or deleted and whether it will run before or after this file operation. The other two allow you to delete file trigger associations and to list the triggers associated with a file.

For more details, refer to *File Triggers* in the Programming in DataBasic section of the *DataBasic Reference Manual*.

## TCP Connections in DataBasic

This feature allows DataBasic programs to connect to and accept connections from remote systems using raw TCP instead of DDA. This allows connections between Reality and many different types of system; for example:

- other Reality systems;
- web, ftp, telnet and time servers;
- SMTP and POP3 email servers;
- networked applications (written in Java, for example);
- other MultiValue systems that support raw TCP;
- XML applications;
- SOAP processes (using XML technology).

For details, refer to the descriptions of the DataBasic *CONNECT* and *ACCEPT* statements.

## Pseudo Floppy

The format used for Reality tape images is different to the pseudo-floppy (.vtf) format used by other MultiValue systems. Two new verbs, *FDISCTOTAPE* and *TAPETOFDISC*, allow you to transfer data between Reality and other MultiValue systems by converting Reality tape images into MultiValue pseudo-floppy images and vice versa.

## Additional ACCOUNT-RESTORE Options

Two additional *ACCOUNT-RESTORE* options are provided to simplify restoring accounts onto systems with a frame size smaller or larger than the original.

## SYSTEM Statement

This new DataBasic statement provides an alternative to using the *ASSIGN statement* for changing system elements whose values can be retrieved using the *SYSTEM function*. Refer to the *DataBasic Reference Manual* for details.

## SP-ASSIGN

By default, the Reality *SP-ASSIGN* command will close any open print jobs. This behaviour can be changed by calling the *SET-OPTION* command with the **SPASSIGN**

option, so that open print jobs will only be closed if SP-ASSIGN is called with no parameters.

## Rapid Recovery File System

This feature provides an additional resilience option. When a database is configured for *Rapid Recovery*, all changes to the database's structure are logged so that it is possible to return the database to a usable state within minutes of restarting after a system failure. There is no need to restore the latest backup from tape. Transaction logs can then be rolled forward and the database brought back into use.

**Note:** This feature is only available on partition databases. It is therefore not available on some existing UNIX databases.

The Rapid Recovery file system is described in greater detail in Chapter 6.

## Compressed Tape Image

This feature allows Reality data to be saved to tape in compressed format, with a choice of compression levels.

The required compression level can be specified in three ways:

1. By setting a database configuration parameter.
2. By setting an operating system environment variable. This overrides any database configuration parameter settings.
3. By modifying the path to the tape image. This can be done in the database configuration file, or by using the T-DEVICE command at TCL. Specifying the level in this way overrides any default set with the previous two methods.

For details of how to set the compression level, refer to *Tape Images* in the Tape Operation and Commands section of the *Reality Reference Manual, Volume 2: Operation*.

The default is no compression, for compatibility with older versions of Reality. Note, however, that a compressed tape image cannot be read by versions of Reality earlier than V9.1. Reality V9.1 and later can read any tape image, whatever the compression level.

The Compressed Tape Image feature is described in greater detail in Chapter 7.

## Support for Distributed Transactions under MTS/COM+

If an application accessing Reality via the SQL/ODBC interface is running in a Microsoft MTS/COM+ environment, it may be using *distributed transactions*. These transactions are fully supported by Reality, using both ODBC and XA interfaces.

Distributed Transactions are described in greater detail in Chapter 8.

## Other New Features

### Large Databases

The maximum database size has been increased from 256 gigabytes to 2 terabytes.

### Networking

UNIX-Connect is now available on Linux. The following features are therefore also now available:

- Remote Tape.
- UNIX-Connect.
- Remote file access.
- DDA terminal interface.
- Remote client server.
- PLID handling.
- Remote Basic.
- RealWeb.
- SQL (ODBC and JDBC).
- Failsafe
- Heartbeat.

### RealEdit

RealEdit is a new Reality editor running under Windows. You can use RealEdit to modify any item in the database to which you have access. It can create and/or modify DataBasic programs, Procs, data file items, and file dictionary items. The only items you cannot edit with RealEdit are cataloged DataBasic programs and other binary format files.

RealEdit is similar in operation to other Windows editors. However, it also allows you to perform Reality-specific operations such as compiling and cataloging DataBasic programs, and viewing included code.

---

## **Chapter 3**

# **GUI Administration**

Via a simple-to-use graphical interface, GUI Administration supports many of the Administrative tasks that can currently be performed from green-screen terminal connections to Reality systems. Enhanced functionality is provided to suit the Client GUI view of the world including maintenance and housekeeping tasks to ensure that Reality systems are internally consistent and coherent.

## GUI Administration

GUI Administration is a suite of programs and files that enable Reality-related administrative tasks to be performed on one or more computer systems from a client graphical user interface running on the same, or different, computer system.

The client component of GUI Administration, referred to hereafter as the *Client GUI*, is deployed to client platforms and allows users to take administrative control of various Reality systems that are visible across the network, subject to the appropriate security checks being satisfied.

Some major features offered by the Client GUI are:

- Presents comprehensive Tree model of Reality systems and databases
- Displays sets of tabbed Panes for tasks relevant to current Tree-node
- Offers consistent, but configurable, 'look and feel' throughout the Client GUI.
- Extensive use of context-sensitive and embedded Help facilities
- Supports and extends existing functionality offered by green-screen equivalents.
- Utilises Wizards to guide users through complex or unfamiliar operations.
- Imposes logon security, on a system-by-system basis, and limits the user's actions and their view of the system, according to their specified privilege level.

The administrator who has been entered as the initial user at installation must first set up more users to administer the system, then select the databases to be part of the network to be administered and authorise the relevant users to administer the databases. The tree structure visible to each user is tailored to their security profile. If a system is not currently available on the network it is indicated in red.

Via the tree-structure the user is able to:

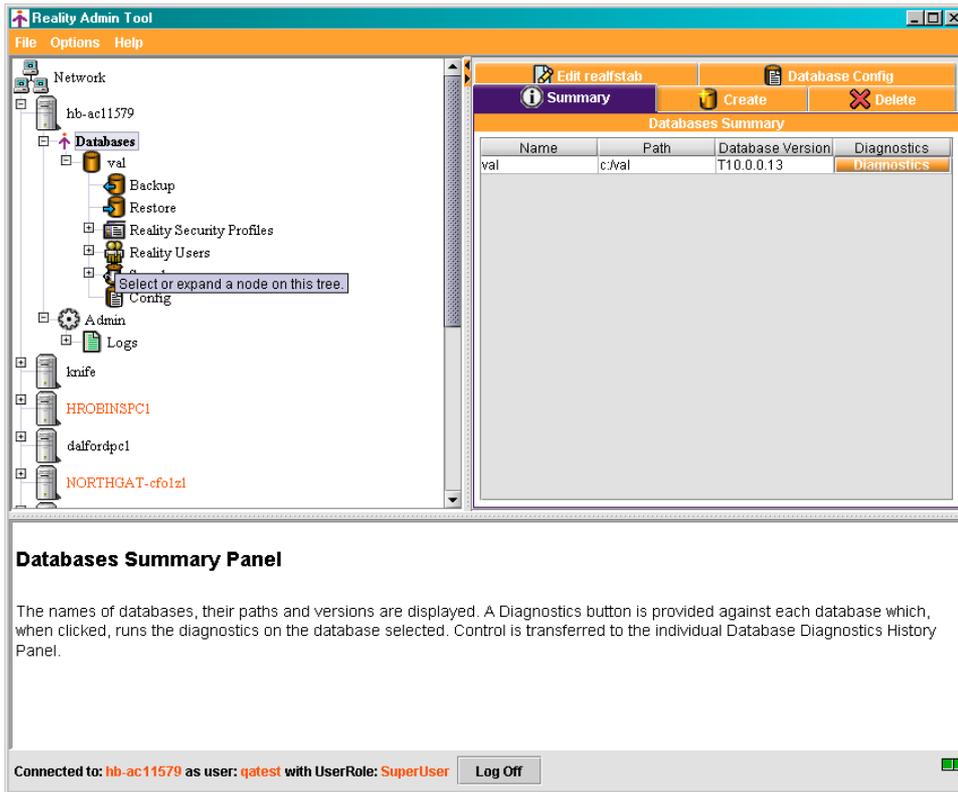
- Set up and maintain databases including:
  - Security profiles.
  - Users.

- Backup & restore.
- Administer the system
  - Add/remove system users.
  - Server administration.
  - Thread management.
  - Set up and administer user access to databases through the GUI Administration tool.

### **Tree Structure**

The system information is presented in a System Tree structure. If the system name is red then the system is not currently connected to the network. If the name is black the system exists on the network but may or may not have a GUI Administration (RAJI) server running. The server must be running for you to access the databases and admin tasks.

Clicking on the + sign preceding the system name expands a structure of Databases and Admin. Clicking on further + signs displays the available databases with associated GUI administration tasks, such as Backup and Restore, setting up Roles and Security Profiles, and the system administration setup tasks.



## Features

Features of Reality GUI Administration which are available at all times:

- A status bar at the base of the screen which displays your log on details, and a Log Off button.
- A File Menu allowing you to Print the tab/screen in focus or Exit.
- An Options Menu with tick boxes to allow you to choose to display the help panel at the base of the GUI Admin panel and/or tooltips on many of the fields requiring input, as in the diagram above. The tooltips (displayed when hovering over the field) may provide sufficient information for you to enter data. If not, then the help provides more details. Links to further background information, including the Reality on-line documentation, are supplied where necessary.
- A Preferences screen allows you to set up, view and change the paths from the client to where the Reality on-line documentation and the browser are stored, and

the location of the list of systems to populate the viewable network. These should be set up before logging in.

- A Help panel (if selected in Preferences) documenting the pane of the GUI Admin Client that you are running.

## Logging In

The System Login dialog is displayed when you attempt to access a system on the Systems Tree by double-clicking a system icon or by attempting to expand a system icon. This dialog will also be presented if your security credentials are revoked while you are within one of the system's sub-nodes, the most likely cause being that you have been timed-out due to inactivity.

You must enter your User name and Password. These fields are initialised with no content the first time a system is accessed during a session of the Client GUI, but thereafter the Username field retains the name of the last submitted user for that particular system. Once both fields have content, pressing Proceed attempts to authenticate your credentials.

When you are successfully authenticated, you must choose which user role you wish to adopt within this system, which determines which tree nodes and panes are accessible. If you have only been assigned one user role, you will be logged on using this role.

## Setting Up and Maintaining Databases

A summary of existing databases and their current status is displayed and diagnostics can also be run on individual databases to check whether the database is clean or needs checking. The database instance summary shows information that allows the administrator to see if there are any potential problems.

The Create Database pane has advanced options which echo those available with the operating system **mksbase** command.

Having created a database you must create Database Access Files (DAFs) from the Admin/Database Access pane to enable the new database to be administered from the GUI Client, and then use Database Users to enable specified users to access the database.

## Backup and Restore

These panes replace the ACCOUNT-SAVE, FILESAVE and DBSAVE with the ACCOUNT-RESTORE, DBSAVE (with restore) and Multiple account restore

## **Users**

A Users Summary panel displays information on all of the users of the selected database. The User Instance summary displays information about the selected user, while the Modify panes include all the fields on the SSM menu, Option 2.

## **Security Profiles**

Replaces Option 3 of the SSM menu.

---

## Chapter 4

# Foreign Database Support

This feature provides Reality with access to data held on SQL-based foreign databases (currently Oracle or SQL Server). There are two ways of doing this, implemented as new Reality file types:

- Reality-specific storage format. In this format, the foreign database is set up to emulate Reality files, thus allowing Reality applications to store their data in the foreign database.
- Data exchange storage format (SQL View). This provides Reality with a view of data held in the foreign database. The data remains in the foreign database format and therefore only limited access is possible from Reality.

Foreign database support requires a working ODBC installation, appropriate ODBC driver(s) and Data Source Definition(s) on the Reality system.

Support for foreign database files and SQL View is not available on AIX and HP platforms.

## Summary of Reality File Types

Reality supports the following types of file:

- **Normal.** An ordinary Reality file on the local database. A normal file has the basic structure described in *File and Item Structures* in the *Reality Users' Reference Manual, Volume 1: General*. Normal files are backed up via the usual Reality save and restore procedures.
- **Foreign Database.** This is a Reality file which has nearly all the characteristics of a Normal file, with the structure described in *File and Item Structures*, but which is located on a foreign (SQL-based) database. You can create, update and manage Foreign Database files in the same way as Normal files. Multi-values and binary items are supported. The files also support indexing and Transaction Handling.

It is unlikely that you would want to back up Foreign Database files using the Reality save commands. If a Reality application is storing its data on a foreign database, it is recommended that this data is secured by saving the SQL database in whatever way is appropriate.

- **SQL View.** This is a Reality file that provides a view of an existing SQL table (or SQL view) on a foreign database. A table, or view, is mapped to the Reality file; each row in that table becomes an item in the Reality file and each column within that row is an attribute in the item.

As the data is stored in a form compatible with the foreign database's native applications, restrictions are imposed on Reality applications writing such data. There is no support for multi-values, indexing or Transaction Handling.

SQL View files are primarily used for read access, but it is possible to update the external table data if you know the exact structure and controls. If you save an SQL View file, you are saving the file definition item, but not the external data, which is secured on the foreign database.

- **Directory View.** A Reality file that provides a view of a directory in the host system (UNIX or Windows). A directory is mapped to the Reality file; each file in the directory appears as an item in that file.

Directory view files are primarily aimed at text file manipulation. A Directory view file contains Reality items for regular system text files within the referenced directory.

Non-regular files - such as sub-directories, pipes and devices – are not visible as items.

The LISTFILES command indicates the different types of file on a particular account.

## LISTFILES

### Purpose

Lists all dictionaries, data sections and index sections defined from a particular account or specified dictionary.

### Command Class

Cataloged DataBasic program

### Syntax

**LISTFILES** {*file-specifier*} { (*options*) }

### Syntax Elements

*file-specifier* is a file, other than the MD, for which the data sections will be listed. If no *file-specifier* is specified, LISTFILES defaults to the current MD, and each file and data section on the account is listed.

The *file-specifier* can be the name of another account if the MD contains a Q-pointer to the other account. If *file-specifier* is the name of a Q-pointer to another account, each file and data section on that other account are listed.

### Options

**P** sends the listing to the printer.

### Report Listing

Each dictionary name is listed with associated data sections listed below, indented by one space. Indexes are listed below associated data sections, indented by two spaces.

### Report Headings

The following information is displayed:

File name            Name of each file.

Type                Type of D/CODE.

Ftype	Type of file This is defined by two characters; a letter and a number. The letter can be:  A A clean log binary data section.  B A byte stream file. (A 'normal' Reality file.)  C A clean log user view data section.  D A directory view.  F A file on a foreign database.  G An SQL view.  The number can be:  1 Master Dictionary  2 File Dictionary  3 Data Section
Mod	Modulo
Sep	Separation
Just	Type of justification (L or R)
Len	Number of columns for output

### Listing Files from Another Account

If the MD of the account you are currently logged onto contains a Q-pointer to another account, the account synonym item-id can be given as the file-name. This then lists the files on the other account.

For example, you are logged on to the INVENTORY account and you want to list the files on the PERSONNEL account. The MD of INVENTORY has a Q-pointer as follows:

```
PERSONNEL
0001 Q
0002 PERSONNEL
```

Then enter:

```
:LISTFILES PERSONNEL
```

The PERSONNEL account files are then listed.

**Example of Report**

File definition items in file: TESTACC							Page 1
File name	Type	Ftype	Mod	Sep	Just	Len	
TAB2	D	B2	1	1	L	10	
TAB2	DY	G3	dbora,bob	TAB2	L	10	
TEMP	D	B2	1	1	L	10	
TEMP	DY	D3	/Temp				
LOCAL	DL	B2	1	1	L	10	
DATA2	DL	B3	11	1	L	10	
LOCAL	DL	B3	11	1	L	10	
X1	D	B4	7	1	L	10	
ORACLE	DL	F2	dbora,bob		L	10	
ORA2	DL	F3	dbora,bob		L	10	
ORACLE	DL	F3	dbora,bob		L	10	
X1	D	F4	dbora,bob		L	10	

In this report:

LOCAL is a local file with two data sections and one index.

ORACLE is a Foreign Database file with two data sections and one index (all stored on an Oracle database called 'dbora')

TEMP is a Directory-View file of directory '/Temp'.

TAB2 is a Sql-View file of Table 'TAB2' on database 'dbora'.

## Foreign Database Files (Reality-specific Storage Format)

The foreign database is set up to emulate Reality files, thus allowing Reality applications to store their data in the foreign database. The Reality account is set up in a local database, with some or all of the files held on a remote foreign database. The file definition item contains details of the file location.

You can create, update and manage the files on the foreign database files in the same way as local Reality files. Multi-values and binary items are supported. The files also support indexing and Transaction Handling.

To create a Reality file on a foreign database, you should use the FDB-SET command to specify details of the foreign database and then use CREATE-FILE to create the file. The FDB-SHOW command lets you display the current foreign database setting. FDB-CLEAR clears the foreign database setting, so that subsequent uses of CREATE-FILE create files on the local Reality database.

### File Definition Item

A file definition item, located in an account's MD, defines the location and characteristics of a file. It points to the file dictionary, giving system information in attribute 2, which specifies the location of the file in the host system, and file creation parameters in attribute 3. It also defines retrieval and update lock codes in attributes 5 and 6, and English formatting characteristics in attributes 7 to 11.

### Item Format

item-id	File-name
---------	-----------

001	D {L R} {X Y}
-----	---------------

where:

'D' marks a Definition Item.

'L' marks that the file is to be logged by Transaction Logging.

'R' indicates that the file is not logged by Transaction Logging, but that its data will be recovered during the automatic recovery of a database configured for Rapid Recovery.

(For more information on the logging status of files, please refer to the [Reality Resilience Reference Manual](#).)

'X' marks that the file is to be ignored by the SAVE command. Use of the O option in the SAVE verb overrides the 'X' code.

'Y' specifies that during the execution of the SAVE verb, no data, except for D-pointers (data level descriptors) is saved to tape. This also applies to FILE-SAVE and ACCOUNT-SAVE which use the SAVE verb. Use of the O option in the SAVE verb overrides the 'Y' code.

002

**On a Filestore Database:**

UNIX path name of File Dictionary

**OR, on a Partition Database:**

*start, id, scatter*

where,

*start* is the logical block no. of the start of file.

*id* is the reuse identifier.

*scatter* is the number of blocks in scatter map.

**OR, for a file on a Foreign Database:**

*n;TableName*

where,

*n* is a number indicating which version of the access method created the file.

*TableName* is the name of the SQL table on the foreign database created to hold this Reality file.

003

**For a file on the Local Database:**

File creation parameters - Filetype (B2) modulo, 1.

**OR, for a file on a Foreign Database:**

$F_n$  *DSN*, {*Userid*}, {*Passwd*}

where

*F* indicates the foreign database access method.

*n* is a number indicating the level of the file in the Reality file system, for example, 3 for a Data section.

*DSN* is the ODBC Data Source Name for the foreign database. The Data Source Name may itself include a user-id and password. For information on how to set up Data Source Names, refer to the documentation for the ODBC installation on your system.

*Userid* is the user id for connecting to the foreign database.

*Passwd* password associated with this userid (encrypted).

004 Null

005 Retrieval lock code(s)

006 Update lock code(s)

**Note:** For a description of lock code operation, refer to [Reality Reference Manual Volume 3](#).

007 Optional conversion codes for English to display item-ids.

008 Optional V (sublist) code for English.

**Note:** For a description of English sublists, refer to the [English Reference Manual](#).

009 Contains a code specifying the type of alignment to be used by English. A code is mandatory and must be one of the following:

- L Left aligned.
- R Right aligned.
- T Text aligned. Text wraps at word boundaries.
- U Unlimited.

**Note:** For more details, refer to the topic [Data Definition Item](#).

- 010 Maximum width of item-id column (default = 10).
- 011 Not used and reserved.
- 012 Null.
- 013 Reallocation parameters. At the next restore from a FILE-SAVE, the File Dictionary is allocated the new modulo and separation parameters specified here.  
  
The format of this specification is  $(m,s)$  expressed in decimal integer format where  $m$  is the new modulo (separation  $(s)$  is always '1' on the current version of Reality).
- 014 - 020 Reserved.

**Note:** If a file has associated indexes, attributes 2, 3 and 13 are multivalued, with the second and subsequent multivalues containing the corresponding information for the indexes. Attribute 4 is also multivalued with the names of the indexes in the second and subsequent multivalues.

## FDB-CLEAR

### Purpose

Used following the execution of FDB-SET to clear the options set by that command. The functionality of the CREATE-FILE command and of restore commands reverts to normal.

### Command Class

TCL-I verb

## Syntax

### FDB-CLEAR

## Example

```
FDB-CLEAR
```

```
[5425] Foreign Database not active
```

## FDB-SET

### Purpose

Changes the functionality of the [CREATE-FILE](#) and restore commands so that files are created/restored on the specified foreign database. This changed functionality of CREATE-FILE remains in place throughout the current logon session, or until the [FDB-CLEAR](#) command is executed.

### Command Class

TCL-I verb

## Syntax

```
FDB-SET database, {user}, {password}
```

## Syntax Elements

<i>database</i>	is the ODBC Data Source Name (DSN) for the foreign database. The DSN may itself include a user-id, and password if required, for logging on to the database. In this case, you will not need to enter them as independent parameters. For information on how to set up DSNs, refer to the documentation for the ODBC installation on your Reality system.
<i>user</i>	is the user-id for logging on to the database. (If the database is not made up entirely of Reality files, it may be partitioned so that all Reality files share one user-id.)
<i>password</i>	is the password associated with this user-id.

## Example

```
FDB-SET FINANCE, REALITY, WELCOME
```

```
[5426] Foreign Database active: Dsn 'FINANCE', User 'REALITY'
```

## FDB-SHOW

### Purpose

Displays the current foreign database setting (and therefore the current functionality of the CREATE-FILE and restore commands).

### Command Class

TCL-I verb

### Syntax

**FDB-SHOW**

### Example

```
FDB-SHOW
```

```
Foreign Database active: Dsn=dbora, User=bob
```

## Saving and Restoring

Although Foreign Database files can be saved using the Reality save commands, it is unlikely that you would want to do this. If a Reality application is storing its data on a foreign database, it is recommended that this data is secured by saving the SQL database in whatever way is appropriate.

If you do save a file on a foreign database using a Reality save command, the File Definition Item is saved with no amendments, so that when the file is restored it is restored onto that foreign database.

It is possible to restore files saved from a Reality database onto a foreign database. Use the [FDB-SET](#) command to indicate that subsequent uses of restore commands should restore files onto the specified foreign database.

**Note:** Before restoring files onto a foreign database, you must first ensure that the required database is online.

## SQL View Files

These files provide Reality with a view of an existing SQL table (or SQL view) on a foreign database. A table, or view, is mapped to the Reality file; each row in that table becomes an item in the Reality file and each column within that row is an attribute in the item.

As the data is stored in a form compatible with the foreign database's native applications, restrictions are imposed on Reality applications writing such data. There is no support for multi-values, indexing or Transaction Handling.

SQL View files are primarily used for read access, but it is possible to update the external table data if you know the exact structure and controls. If you save an SQL View file, you are saving the file definition item, but not the external data, which is secured on the foreign database.

You create an SQL View file using the SQL-VIEW command.

### SQL-VIEW

#### Purpose

Creates a Reality file that provides a view of an existing SQL table (or SQL view) on a foreign database. This command does not create a new table. The DICT section resides in the local Reality database; the Data section(s) are located on the foreign database.

#### Syntax

```
SQL-VIEW file-name database, {user}, {password} Table PkCols DataCols {{(K)}
```

#### Syntax Elements

<i>file-name</i>	is the name of the SQL view file you want to create.
<i>database</i>	is the ODBC Data Source Name for the foreign database. The DSN may itself include a user-id, and password if required, for logging on to the database. If this is the case, you will not need to supply these as independent parameters. For information on how to set up DSNs, refer to the documentation for the ODBC installation on your Reality system.
<i>user</i>	is the user-id for logging on to the database.

<i>password</i>	is the password associated with this user-id.
<i>Table</i>	is the name of the SQL table or view for which you want to create the SQL view file.
<i>PkCols</i>	is the name, or names, of the Primary Key column(s) on the SQL table or view, separated by commas if necessary. Where a table has a single Primary Key column, the Reality item-id maps to this column and is used directly to identify an SQL row. Where a table has multiple Primary Key columns, the Reality item-id comprises the same number of parts and these are used to identify the matching SQL row. The delimiter used to separate the different parts of the item-id is '\' by default. This can be changed by specifying the (K) option, which causes SQL-VIEW to prompt for the Key Separator character.
<i>DataCols</i>	are the data column names in the SQL table or view, separated by commas.

**Option**

K	Causes the command to prompt for a Key Separator character to be used in place of '\'. You can specify any character that is not a Reality system delimiter and which does not appear in the data to be viewed.
---	---

**Restrictions**

As the data is stored in a form compatible with the foreign database's native applications, restrictions are imposed on Reality applications writing such data. There is no support for multi-values, indexing or transaction management.

The foreign database will impose strict control over the type and size of data that may be stored in each column. A Reality application using SQL view files must be aware of the format of the external table data and must keep within these controls. SQL view files may not be updateable, depending on the view definition and the capabilities of the foreign database.

**Comments**

An SQL view file may be based on an SQL table, or on a view definition in the foreign database. A view can be used to filter the data available to Reality, or to combine data from more than one table into a single file view.

---

## Chapter 5

# MultiValue Compatibility

Reality V10.0 has been further enhanced to improve compatibility with other MultiValue systems. The following new features are provided:

- File Triggers.
- TCP Connections in DataBasic.
- Pseudo Floppy support.
- Enhancements to SP-ASSIGN.
- Additional ACCOUNT-RESTORE options.
- SYSTEM statement.

## File Triggers

A file trigger is a cataloged subroutine that is called automatically whenever an item is written to or deleted from a particular file. A trigger can be set to run before or after an item is written and before or after an item is deleted.

Triggers that run before file operations are mainly used to validate the attempted change to the database against user-defined constraints, or “business rules”, and allow the change only if the constraint is satisfied.

Triggers that run after file operations are normally used to create audit trails and other transaction logs.

All types of trigger can be used to create relationships between files, to ensure that whenever one file is updated, another related file is also updated.

**Note:** Triggers are only run when individual items are written to or deleted. They do not run when a complete file is cleared (for example, with the **CLEARFILE** statement).

### How to Write a Trigger Routine

A file trigger must be written as an external subroutine that accepts a single parameter. When the trigger is called by the system, the contents of this parameter depends on the type of trigger:

**Pre-write triggers** The parameter contains the item that is to be written. The data to be written to the item can be modified by changing the contents of this parameter. You can also prevent the item being written by calling the **INPUTERROR** statement.

**Post-write triggers** The parameter contains the data that was written to the item.

**Delete triggers** The parameter always contains a null string.

Within the trigger subroutine, you can access other information about the file and the item by calling **ACCESS** function (see page 5-32). This accepts a data-element number that specifies the type of information you require, as follows:

1 A reference to the trigger file.

- 2 A reference to the dictionary of the trigger file. If the trigger file is a dictionary, this is the same as ACCESS(1).

These two elements allow you to access other items in the trigger file by passing the file reference to I/O statements such as **READ** and **WRITE**.

- 3 The item body. Null if a delete operation. This is similar to the contents of the trigger parameter, but always returns the item data as passed to the trigger; that is, before any changes made by the trigger. Note that in a **POST-WRITE** trigger, this element will contain the data that was written to the item.

- 10 The id of the item being written or deleted.

- 11 The file name in the form **{DICT} {/account/}filename{,data-section-name}**.

- 12 True if the trigger is of type **PRE-DELETE** or **POST-DELETE**.

- 16 True if the item does not exist; false otherwise. Its value therefore depends on the type of trigger:

**PRE-WRITE** True if a new item is being created; false if an existing item is being updated.

**PRE-DELETE** Normally false, but true if the user is attempting to delete a non-existent item (if the item does not exist, no action is taken, but any triggers still run).

**POST-DELETE** Always true.

**POST-WRITE** Always false.

Note that ACCESS(16) checks whether the item exists each time it is called.

- 20 This element is only valid in a **POST-WRITE** trigger – if true, it indicates that the item was modified by the **PRE-WRITE** trigger; if false, the item was written without modification. Note that if you need to compare the original item with that written to the file, you will have to save the original to a variable in a named COMMON area from within the **PRE-WRITE** trigger.

In **PRE-WRITE**, **PRE-DELETE** and **POST-DELETE** triggers this element is always false.

## Debugging Triggers

### Global DEBUG Options

Three options, set with the **SET-OPTION** verb or from within the DataBasic debugger, allow you to specify that DataBasic programs will enter the debugger on encountering a **DEBUG** statement, or if a warning is generated:

**DB.DEBUG** Causes any DataBasic program initiated by the user to enter the DataBasic symbolic debugger on executing **DEBUG** statements within the program. This is similar to starting the program with the **DEBUG** command, but can be used to debug programs called from **PERFORM** statements and from **Procs**.

Programs initiated by the user are those that are started directly or indirectly from **TCL**. They include those initiated by a **Proc** or other program that was itself started directly or indirectly from **TCL**.

**EB.DEBUG** Causes any DataBasic program run from External Basic to enter the DataBasic symbolic debugger on executing **DEBUG** statements within the program. Programs run from External Basic include file triggers and **RealWeb** subroutines; you should set **EB.DEBUG** before starting to debug these types of routine.

### **FATAL.WARNINGS**

Causes all warning messages generated by DataBasic programs to be treated as fatal errors. Breaks to the DataBasic debugger to allow determination of error and possible recovery. Similar to starting the program with the **F** option.

These options can be cleared with the **CLEAR-OPTION** verb or from within the DataBasic debugger.

### Debugger Commands

New debugger commands and additional options to the **SET-OPTION** command make it possible to debug triggers:

**@\*** This command toggles the appropriate global **DEBUG** option (**DB.DEBUG** or **EB.DEBUG**), depending on whether the program being debugged was entered from the **TCL** prompt or was called from External Basic. Equivalent to calling the **SET-OPTION** or **CLEAR-OPTION** verb.

- M** Toggles the option that causes a break whenever a **CALL** or **RETURN** statement is encountered; that is, each time your program calls or returns from an external subroutine.
- WF** Toggles the option that treats warning messages as fatal errors. The effect is limited to the currently running program.
- WF\*** Toggles the global **FATAL.WARNINGS** option (equivalent to calling the **SET-OPTION** or **CLEAR-OPTION** verb).
- WS** The WF command toggles the option that suppresses run-time warning messages (normally enabled by running your program with the **S** option).

Refer to page 5-42 for more details.

**Note:** These debugger commands also make it easier to debug DataBasic programs called from PERFORM statement and Procs.

### Debugger Prompt

If your program has been called from a **PERFORM** statement, the debugger prompt shows the level of nesting. This can help you keep track of which of a number of different programs or routines you are currently debugging. It is particularly useful when debugging file triggers.

File triggers are always at nesting level 1 or above. For example, if a program called from TCL causes a trigger to run, the trigger will be at nesting level 1. If that trigger then carries out an action that causes a second trigger to run, the second trigger will be at nesting level 2 and the debugger prompt will appear as follows:

{2}\*

### How to Associate a Trigger with a File

Once you have written your trigger, you must associate it with the appropriate Reality file using the **CREATE-TRIGGER** command (page 5-25). You must specify the name of the file, the name of the trigger subroutine and the type of trigger required (PRE-WRITE, POST-WRITE, PRE-DELETE or POST-DELETE). Note that the trigger subroutine must be cataloged in the master dictionary of the account containing the file. A trigger can be associated with a file data section, a file dictionary or an account's master dictionary.

The **DELETE-TRIGGER** command (page 5-26) allows you to remove trigger associations. In this case you need only specify the file name and the trigger type.

**LIST-TRIGGERS** (page 5-28) lists the triggers associated with a file.

---

### **Caution**

When associating a trigger with a file, you need to be aware of the effects of any other triggers that might run as a consequence.

---

#### **Notes:**

- It is strongly recommended that triggers should not be associated with system files.
- When accessing files in a different account on the same database (either by defined or direct Q-pointers), the trigger subroutine must exist in the *current* master dictionary.
- When accessing files in a different database, both the trigger definition and trigger subroutine must reside in the remote database where the file resides, not in the local database.
- If a file is open, changes to its associated triggers will not take effect until it is closed and re-opened.
- Only D-pointers can have trigger definitions; Q-pointers will use the definition in the target D-pointer.
- The first time a trigger is used after you have logged on or associated it with a file, it will be copied into a local cache. Subsequent calls to the trigger will therefore be much quicker because it will not be necessary to fetch the trigger from disk. The cache will be cleared when you log off, log to another account or catalog any DataBasic program.

If the cataloged version is changed, only those processes that do not have the trigger cached will use the new version when the trigger next runs. Note that because the local cache is cleared when a DataBasic program is catalogued, the user that catalogs the new version of the trigger will use the new version immediately.

## Commands that might run Triggers

When running the following commands, you should be aware of the effects of any file triggers that might be run as consequence (the list is not exhaustive).

### TCL Commands

<b>BASIC (R)</b> (trigger on MD)	<b>BASIC</b> (trigger on data section)
<b>BLIST (U)</b> (trigger on data section)	<b>CATALOG</b> (trigger on MD)
<b>COPY</b> (trigger on destination file)	<b>CREATE-FILE</b> (trigger on MD)
<b>CREATE-INDEX</b> (trigger on dictionary)	<b>DB</b>
<b>DECAT</b> (trigger on MD)	<b>DELETE</b>
<b>DELETE-FILE</b> (trigger on MD)	<b>DELETE-INDEX</b> (trigger on DICT)
<b>DIR-VIEW</b> (trigger on MD)	<b>ED{IT}</b> (see EDITOR Commands below)
<b>EDELETE</b>	<b>MAKE-SPECIAL</b> (trigger on MD)
<b>MOVE-FILE</b> (trigger on dictionary or MD)	<b>NEW-COPY-LIST</b>
<b>NEW-SAVE-LIST</b>	<b>NEW-SORT-LIST</b>
<b>PQ-COMPILE</b>	<b>REFORMAT</b>
<b>RENAME-FILE</b> (trigger on MD)	<b>RESIZE-FILE</b> (trigger on dictionary)
<b>SE{D{IT}}</b> (see Screen Editor Commands below)	<b>SET-FILE</b> (trigger on MD)
<b>SP-COPY</b>	<b>SQL</b> (update commands)
<b>SQL-VIEW</b> (trigger on MD)	<b>SREFORMAT</b>
<b>T-LOAD</b>	

### Proc Commands

<b>F-DELETE</b>	<b>F-WRITE</b>
-----------------	----------------

### EDITOR Commands

<b>FD</b>	<b>FI</b>	<b>FS</b>
-----------	-----------	-----------

### Screen Editor Commands

**F**

**DataBasic****DELETE** Statement**MATWRITE** Statement**MATWRITEU** Statement**WRITE** statement**WRITEU** Statement**WRITEV** Statement**WRITEVU** Statement**Examples**

The first example is a subroutine that is intended to be used as a **PRE-WRITE** and **PRE-DELETE** trigger. It performs some validation and also records information about the operation for later use by the second example.

```
SUBROUTINE PRE.TRIG(ITEM)
*
* This routine can be used as Pre-Write and Pre-Delete trigger.
*
* It performs some validation and also records information
* about the operation for later use by the Post Trigger.
*
*-----
*
* Named Common - used to share info between Pre and Post Triggers
*
COMMON /TRIGSTUFF/ T.LOGFILE,T.TYPE,T.SEQ
*
* In case we need to debug this routine
* using SET-OPTION EB.DEBUG
*
DEBUG
CRT "PRE-TRIGGER"
*
* Open Logfile for use by Post Trigger
*
IF UNASSIGNED(T.LOGFILE) THEN
*
* We open the logfile here so that we can abort the operation
* if there is any problem. By the time the Post trigger is called
* the operation has completed and cannot easily be cancelled.
*
OPEN "LOGFILE" TO T.LOGFILE ELSE
  INPUTERR "Cannot open logfile - operation aborted"
  RETURN
END
END
*
```

```

* Only users with SYS2 privilege are allowed to update this file
*
PRIV = SYSTEM(28) ;* current system privilege level (0,1,2)
IF (PRIV<2) THEN
  INPUTERR "You do not have permission to update this file"
  RETURN
END
*
* Determine type of operation
*
DELETING = ACCESS(12) ;* True if item being deleted
NEW.ITEM = ACCESS(16) ;* True if new item being written
BEGIN CASE
  CASE DELETING; T.TYPE="DEL";      * delete item
  CASE NEW.ITEM; T.TYPE="NEW";      * write new item
  CASE 1;        T.TYPE="UPD";      * update existing item
END CASE
*
* Check the value of attribute 2 is valid
*
IF NOT(DELETING) THEN
  A2 = ITEM<2>
  IF (A2<0) OR (A2>100) THEN
    INPUTERR "Value out of range"
    RETURN
  END
END
*
* Now, we have successfully opened the logfile
* (if it was not already open)
* verified that the current user is allowed to update this file
* and that the data is correct.
* We have also recorded the type of operation for logging
* in the Post trigger.
*
* Returning without using INPUTERR allows the operation to
complete.
*
RETURN

```

The second example is a **POST-WRITE** and **POST-DELETE** trigger subroutine. It logs the operation using information saved by the previous example.

```

SUBROUTINE POST.TRIG(ITEM)
*
* This routine is a Post-Write and Post-Delete trigger.
*
* It logs the operation using information saved by the Pre Trigger.

```

```
*
*-----
*
* Named Common - used to share info between Pre and Post Triggers
*               and to save a logging sequence number
*
COMMON /TRIGSTUFF/ T.LOGFILE,T.TYPE,T.SEQ
IF UNASSIGNED(T.SEQ) THEN T.SEQ=0
*
* In case we need to debug this routine
* using SET-OPTION EB.DEBUG
*
DEBUG
CRT "POST-TRIGGER"
*
* Construct a unique Log item-id
*
PORT = SYSTEM(18)  ;* current port number
T.SEQ += 1         ;* logging sequence number
*
LOGID = DATE():'~':TIME():'~':PORT:'~':T.SEQ
*
* Construct Log record
*
LOGITEM = ''
LOGITEM<1> = ACCESS(10)  ;* Item-id of item written or deleted
LOGITEM<2> = T.TYPE      ;* Operation type (NEW, UPD or DEL)
*
* Record operation to logfile
*
WRITE LOGITEM ON T.LOGFILE,LOGID ON ERROR
*
* Write to logfile failed, but operation on datafile completed.
* Not much we can do here except shout loudly.
* If using Transaction Boundaries we could use TRANSABORT.
*
CRT
CRT "*** ERROR - Cannot write to logfile"
CRT
END
*
RETURN
```

## Triggers Dos and Don'ts

- It is strongly recommended that you do not associate triggers with system files.

- When you write DataBasic programs or subroutines, always include the **ON ERROR** clause for statements that update files. This is especially important when using triggers, since any failure caused by a trigger aborts the DataBasic program unless the **ON ERROR** clause is present.  
  
**Note:** If the cataloged DataBasic trigger subroutine cannot be found, the trigger will return an error and the update will be aborted. This is because the file would no longer be protected by its business rules and invalid data could be inserted.
- A trigger can invoke another trigger (Reality limits the number of levels you can nest triggers to 256). Be careful to use conditional statements to avoid infinite loops that can be caused by nested triggers.
- **PRE-DELETE** and **POST-DELETE** triggers will run even if the item specified by the user does not exist. You should test element 16 of the **ACCESS** function (page 5-32) for this situation and take appropriate action.
- If you need to pass data between triggers, use variables in a named **COMMON** area.

## TCP Connections in DataBasic

This feature allows DataBasic programs to connect to and accept connections from remote systems using raw TCP instead of DDA. This allows connections between Reality and many different types of system; for example:

- other Reality systems;
- web, ftp, telnet and time servers;
- SMTP and POP3 email servers;
- networked applications (written in Java, for example);
- other MultiValue systems that support raw TCP;
- XML applications;
- SOAP processes (using XML technology).

For details, refer to the descriptions of the DataBasic **CONNECT** and **ACCEPT** statements on pages 5-37 and 5-32 respectively.

### Example Programs

#### TCP Client/Server

The following client and server programs demonstrate how you might use TCP/IP to communicate between two Reality systems.

#### Client Program:

```
* RTCL - Execute remote TCL command
*
* RTCL Host|* TclCommand -- * for localhost"
*
* Connect to XTCL listening on HostName, port 52002, send TCL
* command which XTCL will PERFORM and return the response.
*
* Set environment variable RNWS_LOG_LEVEL in range 0 to 7 for
* tracing. Set environment variable RNWS_LOG_FILE to specify trace
* file.
```

```

*
EQU EDISCONN TO 4235
*
TCL = SENTENCE()
HOST = FIELD(TCL," ",2)
CMD = FIELD(TCL," ",3)
PORT = 52002
*
IF HOST = "" OR HOST = "?" THEN
    PRINT "RTCL Host|* {TclCommand} -- * for localhost"
    STOP
END
*
IF HOST = "*" THEN HOST = "127.0.0.1"
SYS="*TCP*":HOST:";port=":PORT
*
CONNECT SYS TO SESS SETTING ERRNO ELSE
    OPN = "CONNECT"
    GOTO FIN
END
*
ONEOFF = CMD # ""
LOOP
    IF NOT(ONEOFF) THEN INPUT CMD
WHILE CMD # "" DO
    SDAT=CMD
    GOSUB SEND_SDAT
    IF ERRNO THEN GOTO FIN
    GOSUB RECV_RDAT
    IF ERRNO THEN GOTO FIN
    PRINT RDAT<1>
    IF ONEOFF THEN GOTO FIN
REPEAT
*
FIN:
*
IF ERRNO THEN
    IF ERRNO = EDISCONN THEN
        PRINT SYS:" disconnected"
    END ELSE
        PRINT OPN:" (:SYS:) failed, ERRNO=:ERRNO"
    END
END
*
IF NOT(UNASSIGNED(SESS)) THEN
    DISCONNECT SESS SETTING ERRNO ELSE
        PRINT "DISCONNECT (:SYS:) failed, ERRNO=:ERRNO"
    END
END

```

```
    END
*
  IF NOT (UNASSIGNED(LSESS)) THEN
    DISCONNECT LSESS SETTING ERRNO ELSE
      PRINT "DISCONNECT(listener) (:SYS:) failed, ERRNO=:ERRNO"
    END
  END
*
  STOP
*
**
***
SEND_SDAT:
*****
  MSG = "<:SDAT:>"
  SEND MSG TO SESS SETTING ERRNO ELSE
    OPN = "SEND"
  END
  RETURN
*
**
***
RECV_RDAT:
*****
  RDAT = ""
  LOOP
    RECEIVE MSG FROM SESS SETTING ERRNO ELSE
      OPN = "RECEIVE"
      GOTO FIN_RECV_RDAT
    END
    RDAT = RDAT:MSG
    AGAIN = RDAT[-1,1] # ">"
    WHILE AGAIN DO REPEAT
*
  RDAT[-1,1] = ""
  RDAT[1,1] = ""
*
  FIN_RECV_RDAT:
*
  RETURN
*
**
***
END
***
```

**Server Program:**

```

* XTCL - Perform TCL command from remote client
*
* XTCL LocalHostName|* -- * for all local interfaces
*
* Listen on port 52002 for incoming connection, receive and perform
* TCL command and return response to client. Loop back for next
* command or client disconnect. 'RTCL host OFF' will log us off.
*
* Set environment variable RNWS_LOG_LEVEL in range 0 to 7 for
tracing.
* Set environment variable RNWS_LOG_FILE to specify trace file.
*
    EQU EDISCONN TO 4235
*
    LOOPING = 0
    TCL = SENTENCE()
    HOST = FIELD(TCL," ",2)
    PORT = 52002
*
    IF HOST = "" OR HOST = "?" THEN
        PRINT "XTCL LocalHostname|* -- * for all local interfaces"
        STOP
    END
*
    IF HOST = "*" THEN
        HOST = ""
    END
*
    SYS="*TCP*":HOST:";port=":PORT
*
* Activate listening socket
*
    ACCEPT SYS:";listen=1" TO LSESS SETTING ERRNO ELSE
        OPN = "ACCEPT(listen)"
        GOTO FIN
    END
*
* Accept incoming connection
*
    LOOPING = 1
    ERRNO = 0
    LOOP
        ACCEPT SYS TO SESS SETTING ERRNO ELSE
            OPN = "ACCEPT"
            GOTO FIN
        END

```

```

*
  LOOP
    ERRNO = 0
    GOSUB RECV_RDAT
    IF ERRNO = 0 THEN
      IF RDAT = "STOP" THEN
        SDAT = "CLOSING SERVER"
        LOOPING = 0
      END ELSE
        PERFORM RDAT CAPTURING SDAT
      END
    GOSUB SEND_SDAT
  END
  WHILE ERRNO = 0 DO REPEAT
    IF ERRNO = EDISCONN THEN ERRNO=0
  *
  FIN:
  *
    IF ERRNO THEN
      PRINT OPN:" (:SYS:) failed, ERRNO=:ERRNO
    END
  *
    IF NOT(UNASSIGNED(SESS)) THEN
      DISCONNECT SESS SETTING ERRNO ELSE
        PRINT "DISCONNECT (:SYS:) failed, ERRNO=:ERRNO
      END
    END
  *
  WHILE ERRNO = 0 AND LOOPING DO REPEAT
  *
    IF NOT(UNASSIGNED(LSESS)) THEN
      DISCONNECT LSESS SETTING ERRNO ELSE
        PRINT "DISCONNECT(listener) (:SYS:) failed, ERRNO=:ERRNO
      END
    END
  *
  STOP
  *
  **
  ***
  SEND_SDAT:
  *****
    MSG = "<:SDAT:>"
    SEND MSG TO SESS SETTING ERRNO ELSE
      OPN = "SEND"
    END
  RETURN
  *

```

```

**
***
RECV_RDAT:
*****
    RDAT = ""
    LOOP
        RECEIVE MSG FROM SESS SETTING ERRNO ELSE
            OPN = "RECEIVE"
            GOTO FIN_RECV_RDAT
        END
        RDAT = RDAT:MSG
        AGAIN = RDAT[-1,1] # ">"
        WHILE AGAIN DO REPEAT
*
    RDAT[-1,1] = ""
    RDAT[1,1] = ""
*
FIN_RECV_RDAT:
*
    RETURN
*
**
***
END
***

```

## TCP Ping

```

* PING - Reality sockets test program to send/receive to echo server
*
* PING HostName
*
* Provides similar functionality to the UNIX ping utility. A
* connection is established to a remote TCP 'echo' server specified
* by argument 1. A message is sent and echo received, five times.
*
* Set environment variable RNWS_LOG_LEVEL in range 0 to 7 for
* tracing. Set environment variable RNWS_LOG_FILE to specify trace
* file.
*
    TCL = SENTENCE()
    HOST = FIELD(TCL," ",2)
    PORT=7
*
    IF HOST = "" OR HOST = "?" THEN
        PRINT "PING [HostName|IP address]"
        STOP
    END

```

```
*
SYS="*TCP*":HOST:";port=":PORT
*
CONNECT SYS TO SECHO TIMEOUT 1 SETTING ERRNO ELSE
    PRINT "Failed to connect to ":SYS:", error=":ERRNO
    GOTO L_ABORT
END
*
MSG="Data/Basic socket call to echo port"
*
FOR I = 1 TO 5
    SEND MSG TO SECHO SETTING ERRNO ELSE
        PRINT "Failed to send msg to ":SYS:", error=":ERRNO
        GOTO L_ABORT
    END
*
    RECV MSG FROM SECHO TIMEOUT 1 SETTING ERRNO ELSE
        PRINT "Failed to receive msg from ":SYS:", error=":ERRNO
        GOTO L_ABORT
    END
    PRINT "Received:'":MSG:'' from ":HOST
NEXT I
*
L_ABORT:
    IF NOT (UNASSIGNED(SECHO)) THEN
        DISCONNECT SECHO SETTING ERRNO ELSE
            PRINT "Failed to disconnect from ":SYS:", error=":ERRNO
        END
    END
*
STOP
*
END
```

### TCP Email

```
* MAIL - send email
*
* MAIL <To:> <From:> <SmtpServer>
*
* Connect to SMTP service on specified DNS name and pass sender and
* recipient's mail path, viz email addresses, finally send the email
* body.
*
* This program was based on information found in RFC 821, it in no
* way handles every eventuality and is for example purposes only.
*
    EQU EDISCONN TO 4235
```

```

EQU TRACING TO 1
CRLF = CHAR(13):CHAR(10)
*
TCL = SENTENCE()
EML.TO   = FIELD(TCL," ",2)
EML.FROM = FIELD(TCL," ",3)
SMTPHOST = FIELD(TCL," ",4)
PORT = 25
*
IF SMTPHOST = "" THEN
    PRINT "MAIL <To:> <From:> <SmtpServer>"
    STOP
END
*
SYS="*TCP*":SMTPHOST:":port=":PORT
*
* Build email header
*
EML.HEADER = "Subject: TEST Sockets API":CRLF:"From: ":EML.FROM
EML.HEADER = EML.HEADER:CRLF:"To: ":EML.TO
*
PRINT "Enter message text:"
INPUT EML.TEXT
*
CONNECT SYS TO SESS SETTING ERRNO ELSE
    OPN = "CONNECT"
    GOTO FIN
END
*
SDAT = "MAIL FROM:":EML.FROM
GOSUB POST; IF ERRNO THEN GOTO FIN
IF RDAT[1,6] # "250 OK" THEN GOTO PROTOCOL.ERROR
*
SDAT = "RCPT TO:":EML.TO
GOSUB POST; IF ERRNO THEN GOTO FIN
IF RDAT[1,6] # "250 OK" THEN GOTO PROTOCOL.ERROR
*
SDAT = "DATA"
GOSUB POST; IF ERRNO THEN GOTO FIN
IF RDAT[1,3] # "354" THEN GOTO PROTOCOL.ERROR
*
* Build email body - header, blank line, text & a '.' on it's own
line,
* this, with the final CRLF added by POST, terminates the message
body.
*
SDAT = EML.HEADER:CRLF:CRLF:EML.TEXT:CRLF: "."
GOSUB POST; IF ERRNO THEN GOTO FIN

```

```

    IF RDAT[1,6] # "250 OK" THEN GOTO PROTOCOL.ERROR
*
    GOTO FIN
*
PROTOCOL.ERROR:
*
    PRINT "Unexpected response from SMTP server."
    PRINT "SDAT=":SDAT
    PRINT "RDAT=":RDAT
*
FIN:
*
    IF ERRNO THEN
        IF ERRNO = EDISCONN THEN
            PRINT SYS:" disconnected"
        END ELSE
            PRINT OPN:" (:SYS:) failed, ERRNO=:ERRNO"
        END
    END
*
    IF NOT(UNASSIGNED(SESS)) THEN
        DISCONNECT SESS SETTING ERRNO ELSE
            PRINT "DISCONNECT (:SYS:) failed, ERRNO=:ERRNO"
        END
    END
*
    STOP
*
**
***
POST:
*****
    IF TRACING THEN PRINT "POST:":SDAT
    MSG = SDAT:CRLF
    SEND MSG TO SESS SETTING ERRNO THEN
        GOSUB RECV
    END ELSE
        OPN = "SEND"
    END
    RETURN
*
**
***
RECV:
*****
    RDAT = ""
    LOOP
        RECEIVE MSG FROM SESS SETTING ERRNO ELSE

```

```
        OPN = "RECEIVE"
        GOTO FIN_RECV
    END
    RDAT = RDAT:MSG
    AGAIN = RDAT[-2,2] # CRLF
    WHILE AGAIN DO REPEAT
    *
    RDAT[-2,2] = ""
    *
    IF TRACING THEN PRINT "RCV:":RDAT
    IF RDAT[1,3] = "220" THEN
    *   This is the greeting reply from server, is ok, go do another
receive
        GOTO RECV
    END

FIN_RECV:
*
    RETURN
*
**
***
END
***
```

## Pseudo Floppy Support

The format used for Reality tape images is different to the pseudo-floppy (.vtf) format used by other MultiValue systems. Two new verbs, **FDISCTOTAPE** (page 5-26) and **TAPETOFDISC** (page 5-30), allow you to transfer data between Reality and other MultiValue systems by converting Reality tape images into MultiValue pseudo-floppy images and vice versa.

## SP-ASSIGN Enhancements

By default, the Reality **SP-ASSIGN** command will close any open print jobs. This behaviour can now be changed by calling the **SET-OPTION** command with the **SPASSIGN** option, so that open print jobs will only be closed if **SP-ASSIGN** is called with no parameters.

## Additional ACCOUNT-RESTORE Options

Two additional options are provided for the **ACCOUNT-RESTORE** command. These simplify restoring accounts onto systems with a frame size smaller or larger than the original.

- D** Doubles the modulo in the files restored. Use this option if the target system has a smaller frame size.
- H** Halves the modulo in the files restored. Use this option if the target system has a larger frame size.

Note that the new modulus will be only a working approximation to allow the system to work reasonably efficiently. For optimum efficiency, you should resize the files.

## TCL Commands

### CREATE-TRIGGER

Associates a trigger with a Reality file.

#### Syntax

**CREATE-TRIGGER** *file-specifier trigger-name trigger-type*

#### Syntax Elements

*file-specifier* is the name of the file with which the trigger will be associated. This can be a file data section, a file dictionary or an account's master dictionary.

*trigger-name* The name of the trigger subroutine. This must be a cataloged DataBasic subroutine in the MD of the account containing the file.

*trigger-type* The type of trigger – one of the following:

**WRITE or PRE-WRITE**

Run the trigger routine before writing a file item.

**POST-WRITE**

Run the trigger routine after writing a file item.

**DELETE or PRE-DELETE**

Run the trigger routine before deleting an item from the file.

**POST-DELETE**

Run the trigger routine after deleting an item from the file.

#### Restrictions

Requires SYS2 privileges.

#### Comments

The file specified may be a local file (dictionary or data section) or the master dictionary of an account.

When associating a new trigger with a file, any existing trigger of the specified type must first be deleted (see **DELETE-TRIGGER** on page 26).

**Example**

```
:CREATE-TRIGGER TF TRIG1 PRE-WRITE  
[1901] 'PRE-WRITE' trigger added to file 'TF'.
```

**DELETE-TRIGGER**

Deletes one or all of the trigger associations for a Reality file.

**Syntax**

```
DELETE-TRIGGER file-specifier [trigger-type | * ]
```

**Syntax Elements**

*file-specifier* is the name of the file with which the trigger is associated.

*trigger-type* The type of trigger: **WRITE**, **PRE-WRITE**, **POST-WRITE**, **DELETE**, **PRE-DELETE** or **POST-DELETE** (see **CREATE-TRIGGER**).  
\* specifies all triggers.

Note that the keywords **WRITE** and **DELETE** are synonyms for **PRE-WRITE** and **PRE-DELETE** respectively,

**Restrictions**

Requires SYS2 privileges.

**Comments**

The file specified may be a local file (dictionary or data section) or the master dictionary of an account.

**Example**

```
:DELETE-TRIGGER TF *  
[1915] All triggers deleted from file 'TF'.
```

**FDISCTOTAPE****Purpose**

Converts a MultiValue pseudo-floppy image into a Reality tape image.

**Syntax**

```
FDISCTOTAPE file-specifier item-id path {{options}
```

**Syntax Elements**

<i>file-specifier</i>	The name of a Reality binary <b>DIR-VIEW</b> file containing the MultiValue pseudo floppy image.
<i>item-id</i>	The name of the item containing the MultiValue pseudo floppy image.
<i>path</i>	The native directory path into which to save the Reality tape image.

**Note:** You will be prompted for any missing parameters.

**Options**

<b>Cn</b>	Sets the Reality tape compression to level <i>n</i> ; defaults to 0.
<b>S</b>	Suppresses the progress '#' characters.

**Operation**

FDISCTOTAPE reads the MultiValue pseudo floppy image *item-id* from the Reality binary DIR-VIEW file *file-specifier* and converts it to a Reality tape image. The tape image is saved in the native directory defined by *path* and is given the same name as the pseudo floppy image, but with the file extension '.rti' or '.rci'. Compressed tape images have the extension '.rci' and uncompressed images the extension '.rti'.

---

**Caution**

Any existing file with the same name will be overwritten.

---

**Restrictions**

The size of image that can be converted is limited to about 60Mb by the active workspace limit.

**Note:** This restriction is likely to be removed on future releases; please check the Northgate web site for the latest product and documentation updates.

You can increase the active workspace limit by setting the environment variable RWSMAXSIZE. For example, if you need to convert a 200Mb image, you should set RWSMAXSIZE to at least 204800.

- On UNIX, add a line containing the following to the file **.realityrc** in your home directory.

`RWSMAXSIZE=limit`

where *limit* is the required active workspace limit in kilobytes.

Once you have converted your pseudo floppy image, you should return the active workspace limit to its original value by removing `RWSMAXSIZE` from `.realityrc`.

- On Windows, set the environment variable when you log on, as follows:

```
Logon please : username RWSMAXSIZE=limit
```

where *username* is your Reality user name and where *limit* is the required active workspace limit in kilobytes.

In neither case will this change affect other Reality users.

### Example

```
DIR-VIEW HOST-FILES C:\images\tapes (B
FDISCTOTAPE HOST-FILES PFDATA C:\images\tapes (C6
```

Sets up a binary directory view of the PC directory `C:\images\tapes` and then converts the item `PFDATA` into a Reality tape image. The tape image is saved with level 6 compression, in the file `C:\images\tapes\PFDATA.rci`.

## LIST-TRIGGERS

Lists the triggers associated with a Reality file.

### Syntax

```
LIST-TRIGGERS file-specifier
```

### Syntax Elements

*file-specifier* is the name of the file for which to list the triggers.

### Comments

The file specified may be a local file (dictionary or data section) or the master dictionary of an account.

### Example

```
:LIST-TRIGGERS TEST
```

```
File 'TF' has the following triggers:
```

```
PRE-WRITE: T1
```

PRE-DELETE: T2

2 trigger(s) listed.

## TAPETOFDISC

### Purpose

Converts a Reality tape image into a MultiValue pseudo-floppy image.

### Syntax

**TAPETOFDISC** *path* *tape-image* *file-specifier* *{(options)}*

### Syntax Elements

<i>path</i>	The native directory path containing the Reality Tape image.
<i>tape-image</i>	The name of the native file containing the Reality Tape image. This must have a file extension of either '.rci' or '.rti'.
<i>file-specifier</i>	The name of a Reality binary <b>DIR-VIEW</b> file into which to save the MultiValue pseudo floppy image.

**Note:** You will be prompted for any missing parameters.

### Options

**S** Suppresses the progress '#' characters.

### Operation

TAPETOFDISC reads the Reality tape image *tape-image* from the directory *path* and converts it to a MultiValue pseudo floppy image. The pseudo floppy image is saved in the Reality binary DIR-VIEW file *file-specifier* as an item with the same name as the tape image, but without the extension '.rti' or '.rci'.

---

### Caution

Any existing item with the same name will be overwritten.

---

### Restrictions

Only Reality tapes images that have a block size of 500 can be converted to MultiValue pseudo floppy images.

The size of image that can be converted is limited to about 60Mb by the active workspace limit (see **FDISCTOTAPE** on page 5-26 for more details).

**Example**

```
DIR-VIEW HOST-FILES C:\images\tapes (B  
TAPETOFDISC C:\images\tapes PFDATA.rci HOST-FILES
```

Sets up a binary directory view of the PC directory **C:\images\tapes** and then converts the Reality tape image **C:\images\tapes\PFDATA.rci** into a MultiValue pseudo floppy image. The image is saved in the DIR-VIEW file as the item **PFDATA**.

## DataBasic Statements and Functions

### ACCEPT Statement

#### Purpose

To declare the availability of the server to the local session manager, or to accept a connection from a client that has just requested a connection.

#### Syntax

```
ACCEPT accept-string TO session {TIMEOUT minutes}  
{SETTING error} {RETURNING client-id} [THEN statement(s) | ELSE statement(s)]
```

#### Syntax Elements

<i>accept-string</i>	is a string with one of the formats described in the section <b>Accept String</b> .
<i>session</i>	is a variable to hold a "session handle". On return this will contain a value that identifies the connection.
<i>minutes</i>	is an expression giving a timeout value in minutes. The ELSE clause is executed if a connect request is not received and a session established within this time. If a connect request is not received and the TIMEOUT clause is omitted (or <i>minutes</i> = 0), <i>error</i> is set to 4225 and the ELSE clause is taken.
<i>client-id</i>	is a dynamic array variable consisting of two attributes that identify the client program, that is:  <i>client-plid</i> ^ <i>client-system</i> * <i>user-id</i>  where:  <i>client-plid</i> is the PLId of the process running the client program. This forms the first attribute.  <i>client-system</i> is the name of the system running the client program, specified in the client's routing information (ROUTE-FILE in UNIX or the registry in Windows).

---

<i>user-id</i>	is the user-id used to logon the client process. This forms part of the second attribute and is separated from the first part, <i>client-system</i> , by an asterisk (*).
^	represents an attribute mark inserted by entering CTRL+^
<i>error</i>	is a variable that is assigned an error code number according to any errors detected if the ELSE clause is taken. If the ELSE clause is not taken, the value of error is set to 0. The error codes and corresponding messages are given in the <i>DataBasic Reference Manual</i> .
<i>statement(s)</i>	comprises one or more DataBasic statements, executed as part of a THEN or ELSE clause. A statement must be included. The THEN clause is executed if the ACCEPT establishes a session without error, otherwise, the ELSE clause is executed.

**Accept String**

The *accept-string* parameter must be a string with one of the following formats:

- To accept a connection from a Reality client program:

**{\*PTP\*}***server*

where:

**\*PTP\*** specifies that this is a Reality process-to-process connection. Note that, for Reality process-to-process this element is optional.

*server* is the name by which the client knows this PTP server.

- To accept a connection from a remote system using raw TCP/IP:

**\*TCP\****host;port=port{;option}{;option}...*

where:

**\*TCP\*** specifies that this is a raw TCP/IP connection.

*host* is the IP address on which to accept a connection. This can be the IP address of a local network interface, blank to listen on all

local network interfaces, or for local loopback, either the IP address 127.0.0.1 or "localhost".

*port* is the port on *host* from which to accept a connection.

*option* is a name/value pair (separated by an equals sign), specifying an optional parameter to be passed to the host.

Examples:

```
**TCP*152.114.24.126;port=1045;listen=1"
```

Listens on port 1045 of the network interface with IP address 152.114.24.126.

```
**TCP*;port=52002;listen=1"
```

Listens on port 52002 of all local network interfaces.

```
**TCP*localhost;port=2701;listen=1"
```

Listens on local loopback port 2701.

## Operation

### DDA Connections

The ACCEPT statement is used either to declare to the Session Manager that the server is available to any client that might subsequently request connection or it might be used as a reply to accept connection to a client that has just requested connection.

### Raw TCP/IP Connections

When using the ACCEPT statement to accept raw TCP/IP connections, you must first create a listening socket by issuing an ACCEPT with the listen option. For example:

```
ACCEPT "**TCP*152.114.24.126;port=1045;listen=1" TO LISTENSESS ELSE STOP
```

Incoming connections from this host will then be queued. The listen option specifies the size of the queue (note that the operating system may limit on the size of the queue).

Subsequent ACCEPT calls, specifying the same host and port, but without the listen option, can then be used to fetch connection requests from the queue. For example:

```
ACCEPT "**TCP*152.114.24.126;port=1045" TO CONNSESS ELSE STOP
```

When your program has finished with a connection or no longer wishes to accept incoming connections on the specified address, it should issue a disconnect:

DISCONNECT CONNSESS ELSE STOP

## ACCESS Function

### Purpose

To provide access to the current states of various data elements. Can be used only within a file trigger subroutine.

### Syntax

ACCESS(*data-element*)

### Syntax Elements

*data-element* is the number corresponding to the data element to be referenced.

### Operation

These values of *data-element* return the following information:

- 1 A reference to the trigger file.
- 2 If the trigger file is a data section, a reference to the dictionary of the trigger file. If the trigger file is a dictionary, a reference to the trigger file.
- 3 The item body. Null if a delete operation.
- 10 The id of the item being written or deleted.
- 11 The file name in the form {**DICT**} {/account/}filename{,data-section-name}.
- 12 True if a **PRE-DELETE** or **POST-DELETE** trigger.
- 13 Always returns 0.
- 16 True if the item does not exist; false otherwise. Its value therefore depends on the type of trigger:

**PRE-WRITE** True if a new item is being created; false if an existing item is being updated.

**PRE-DELETE** Normally false, but true if the user is attempting to delete a non-existent item (if the item does not exist, no action is taken, but any triggers still run).

**POST-DELETE** Always true.

**POST-WRITE** Always false.

Note that ACCESS(16) checks whether the item exists each time it is called.

20 In a **POST-WRITE** trigger, if true, indicates that the item was modified by the PRE-WRITE trigger; if false, the item was written without modification. Always false in **PRE-WRITE**, **PRE-DELETE** and **POST-DELETE** triggers.

23 The calling environment. Currently always returns 1 (trigger).

### Examples

```
* Named Common - used to share info between Pre and Post Triggers
COMMON /TRIGSTUFF/ T.TYPE

* Determine type of operation
DELETING = ACCESS(12)  ;* True if item being deleted
NEW.ITEM = ACCESS(16) ;* True if new item being written
BEGIN CASE
  CASE DELETING; T.TYPE="DEL";    * delete item
  CASE NEW.ITEM; T.TYPE="NEW";    * write new item
  CASE 1;       T.TYPE="UPD";    * update existing item
END CASE
```

This example calls ACCESS(12) and ACCESS(16) to determine whether an item is being created, updated or deleted, and stores the result in a named common area for use in a subsequent trigger.

```
* Construct Log record
LOGID = DATE():'~':TIME()
LOGITEM = ''
LOGITEM<1> = ACCESS(10) ;* Item-id of item written or deleted
LOGITEM<2> = T.TYPE     ;* Operation type (NEW, UPD or DEL)
*
* Record operation to logfile
*
WRITE LOGITEM ON LOGFILE,LOGID ON ERROR
*
* Write to logfile failed, but operation on datafile completed.
* Not much we can do here except shout loudly.
* If using Transaction Boundaries we could use TRANSABORT.
*
CRT
CRT "*** ERROR - Cannot write to logfile"
```

```
CRT  
END
```

This example calls ACCESS(10) obtain the item-id of the item being written of deleted and then writes this information to a log file.

## CONNECT Statement

### Purpose

To establish a connection between the client program and a server on a local or remote system.

### Syntax

```
CONNECT connect-string TO session {TIMEOUT minutes} {SETTING error}  
[THEN statement(s) | ELSE statement(s)]
```

### Syntax Elements

<i>connect-string</i>	is a string with one of the formats described in the section <a href="#">Connect String</a> . This specifies the protocol to use, the host to which to connect, etc.
<i>session</i>	is a variable to hold a “session handle”. On return this will contain a value that identifies the connection.
<i>minutes</i>	is an expression giving a timeout value in minutes. The ELSE clause is executed if a connect request is not received and a session not established within this time. If the TIMEOUT clause is omitted, the program waits indefinitely for the CONNECT to complete.
<i>error</i>	is a variable that is assigned an error code number if the CONNECT operation fails. If the connection is established successfully, <i>error</i> is set to 0. The error codes and corresponding messages are given in the <i>DataBasic Reference Manual</i> .
<i>statement(s)</i>	is either a THEN or ELSE clause (or both). A statement must be included. The THEN clause is executed if the CONNECT establishes a connection without error. The ELSE clause is executed otherwise.

### Connect String

The *connect-string* parameter specifies the protocol to use, the host to which to connect, etc. The available protocols are:

- [Reality process-to-process](#).

- [TCP/IP](#).

**Reality process-to-process:**

For a Reality process-to-process connection, *connect-string* must be a string with the following format:

```
{*PTP*}{system}^{account{,acct-passwd}}^{server{,server-passwd}}{^Q}
```

where:

<b>*PTP*</b>	specifies that this is a Reality process-to-process connection. Note that, for Reality process-to-process this element is optional.
<i>system</i>	is an entry in the UNIX <b>ROUTE-FILE</b> or Windows registry, that identifies the remote system to connect to. The local database is used if this is omitted.
<b>^</b>	represents an attribute mark (character with decimal value 254, typed as CTRL+^). All except the last of these must be included.
<i>account</i>	is an account on which a server is to be started. Can be omitted if the server should already be running.
<i>acct-passwd</i>	is the password for the named <i>account</i> , if one is required, unless the account is the server's default account in which case the <i>acct-passwd</i> is not required.
<i>server</i>	is a command in the named account's MD that executes a server program or the name of a server program already running on the specified system or database. In the latter case, the name of the server program is that specified as its server-id in the ACCEPT statement it executes.

If the server-id specified in the CONNECT statement exists as a user-id on the remote system (the system to which connection is being connected), then the server-id is used as the user-id to log to and its associated user profile is used to validate the connection.

If the server-id is not defined as a user-id, the local user's profile is checked for a network id (net-id). If a net-id is defined then this is used as the user-id. However, the net-id must also be

defined as a user-id on the remote system otherwise the connection will fail.

If a net-id is not defined on the local system either, then the user-id used on the remote system defaults to the user-id used to logon the process running the client program.

*server-passwd* is required if the server requires a password.

**Q** is the character Q. If it is appended after an attribute mark, queues the connect instead of starting a server.

The connect request is queued until an already-running server with the name specified issues an accept.

For example:

```
"FINANCE" : AM : "SALES-ACCOUNT" : AM : "ORDER-SERVER"
```

#### **TCP/IP:**

For connection to a remote system using raw TCP/IP, *connect-string* must be a string with the following format:

```
*TCP*host;port=port{;option}{;option}...
```

where:

**\*TCP\*** specifies that this is a raw TCP/IP connection.

*host* is the IP address or DNS domain name of the system to which to connect.

*port* is the port on *host* to which to connect.

*option* is a name/value pair (separated by an equals sign), specifying an optional parameter to be passed to the host.

For example:

```
"*TCP*152.114.24.123;port=21;linger=5000"
```

## Operation

The CONNECT statement is executed by a client program to establish a connection with a server program on a local or remote system.

- For Reality process-to-process connections, the connection is made to the system and account identified by the system and account parameters in the *connect-string* parameter. If a program, identified by *server*, is running and has issued an ACCEPT statement to this process, the connection is made. If not, the server is instructed to start the program on the server system.
- For TCP/IP connections, the connection is made to the system identified by the *host* parameter. The port used is specified in the *port* parameter. Connections can be to various kinds of remote system. For example:
  - other Reality systems;
  - web, ftp, telnet and time servers;
  - SMTP and POP3 email servers;
  - networked applications (written in Java, for example);
  - other MultiValue systems that support raw TCP/IP;
  - XML applications;
  - SOAP processes (using XML technology).

Note, however, that the CONNECT statement provides only a *raw* TCP connection. You must implement the protocols needed for communication with these remote systems.

Once a session has been started, either the client or the server can send data, receive data, or terminate the connection.

A program can maintain more than one connection at the same time. Each connection is identified in SEND, RECEIVE, RECWAIT, and DISCONNECT statements by the session handle that is assigned by the CONNECT and ACCEPT statements.

## SYSTEM Statement

### Purpose

To allow the states of various system elements to be changed. An alternative to the **ASSIGN** statement for setting a system element.

### Syntax

**SYSTEM**(*sys-element*) = *value*

### Syntax Elements

*sys-element* is the number corresponding to the system element to be changed.

### System Elements

Only the following system elements can be changed:

- 2 Current page width (numeric).
- 3 Current page length (numeric).
- 5 Current page number (numeric).
- 7 Terminal type (numeric).
- 30 Pagination in effect (numeric).
- 35 Language in use (numeric).
- 37 Thousands separator in use (string).
- 38 Decimal separator in use: comma or period (string).
- 39 Money sign in use (string).

### Examples

```
SYSTEM(5) = 12
```

Assigns the value 12 to system element 5, the current page number.

### See Also

**SYSTEM** Function, **ASSIGN** Statement.

## Debugger Commands

### @

#### Purpose

To inhibit a break if a DEBUG statement is encountered.

#### Syntax

@{\*}

#### Syntax Elements

\* Toggles the appropriate global DEBUG option (**DB.DEBUG** or **EB.DEBUG**), depending on whether the program being debugged was entered from the TCL prompt or was called from External Basic. Equivalent to calling the **SET-OPTION** or **CLEAR-OPTION** verb.

**Note:** Use **SET-OPTION** to set **EB.DEBUG** before starting to debug file triggers.

If this element is omitted, the effect of the @ command is limited to the program currently being debugged.

#### Operation

If a program contains one or more DEBUG statements and the program was run via the DEBUG command, an execution break occurs every time a DEBUG statement is encountered.

The @ command toggles the function of the DEBUG statement. Issuing the @ command one time inhibits breaking. Issuing it a second time turns it back on.

The words ON and OFF are printed next to the @ to indicate the current status of the @ command.

#### Examples

```
*@ ON
```

Indicates that any subsequent DEBUG statements will be ignored.

```
*@ OFF
```

Turns the **@** command off, so a subsequent **DEBUG** statement will cause an execution break.

## **M**

### **Purpose**

The **M** command toggles the option that causes a break whenever a **CALL** or **RETURN** statement is encountered; that is, each time your program calls or returns from an external subroutine.

### **Syntax**

**M**

### **Operation**

The words **ON** and **OFF** are printed next to the **M** to indicate the status of the option.

## **WF**

### **Purpose**

The **WF** command toggles the option that treats warning messages as fatal errors.

### **Syntax**

**WF{\*}**

### **Syntax Elements**

**\*** Toggles the global **FATAL.WARNINGS** option (equivalent to calling the **SET-OPTION** or **CLEAR-OPTION** verb). You should use this option when debugging file triggers.

If this element is omitted, the effect of the **WF** command is limited to the program currently being debugged.

### **Operation**

The option that treats warning messages as fatal errors can be enabled by running your program with the **F** option. The **WF** debugger command toggles this option on and off. The words **ON** and **OFF** are printed next to the **WF** to indicate the status of the option.

## **WS**

### **Purpose**

The WS command toggles the option that suppresses run-time warning messages.

### **Syntax**

**WS**

### **Operation**

The option that suppresses run-time warning messages can be enabled by running your program with the **S** option. The **WS** debugger command toggles this option on and off. The words ON and OFF are printed next to the WS to indicate the status of the option.

---

## Chapter 6

# Rapid Recovery File System

This feature provides an additional resilience option. All changes to a database's structure are logged so that it is possible to return a database to a usable state within minutes of restarting after a system failure. There is no need to restore the latest backup from tape. Transaction logs can then be rolled forward and the database brought back into use.

**Note:** This feature is only available on partition databases. It is therefore not available on some existing UNIX databases.

## Description of Rapid Recovery

### What is Rapid Recovery?

Rapid Recovery is a software facility that enables quick restoration of a database to a valid structure following a system failure. Structural changes to a database are saved to a dedicated log disk so that, if the database becomes corrupted, recent changes can be rolled back until the database is in a consistent state. The images of structural changes are known as Quick images.

Rapid Recovery is normally used together with [Transaction Logging](#) to ensure that both the database structure and the users' data are restored during recovery. However, for purposes of efficiency, it is possible to configure a database (or individual files) to a lower level of resilience, but still maintain recovery of the database structure.

### How Rapid Recovery Works

Each time a structural change is made to a database that has Rapid Recovery enabled, the Rapid Recovery software writes the change to the raw log located on the log disk.

The raw log is a raw partition on the log disk. It acts as a central repository for the Before and After images saved by the Transaction Logging software (see [Raw Log](#)) and for the Quick images logged by the Rapid Recovery software. One raw log partition is shared by all databases on a system.

Rapid Recovery software will save After images of changes to any file that is defined as Recoverable. These images are written to the raw log, as described in [Introduction to Transaction Logging](#) in the *Reality Resilience Manual* but are tagged as 'phantom' to indicate that they should not be copied to the clean log. (If the database is not configured for Transaction Logging, there will be no clean log available.) After images are available on the raw log during automatic database recovery to restore the file's data.

The logging status of files on a Rapid Recovery database can be:

- **Logged:** Before and After images are written to the raw log as well as the Quick images. The After images are then stored in the clean log as an audit trail and for disaster recovery. (This status is only available if the database is also configured for Transaction Logging.)
- **Recoverable:** Before and After images are written to the raw log as well as the Quick images, but the After images are tagged as 'phantom' and are not written to the clean log. This mode may be useful for application level indexes for example.

- **Not Logged:** Only Quick images are written to the raw log. Following automatic recovery, the database will have a consistent structure but there may be data missing from these files. This mode is suitable for scratch files.

## Configuring a Database for Rapid Recovery

To use Rapid Recovery on a particular database it must first be configured. This is carried out from **tlmenu**'s Configuration and Setup menu (Figure 6-1), displayed by selecting option 2 on the main administration menu. A full description of **tlmenu** is provided in the *Reality Resilience Manual*.

```
Transaction Logging Menu System           Mon Jul 31 1999
Database name : dbase2                   Host name : crime
State          : Transaction Handling/Logging not enabled

Configuration and Setup
=====

1. Define/Redefine the Database Configuration
2. Start Transaction Logging
3. Stop Transaction Logging
4. Configure the Transaction logging Status Monitor

Enter option (1-4) : _
```

**Figure 6-1. Transaction Logging Configuration and Setup Menu**

1. Select option 1 on the Configuration and Setup menu. A message is then displayed describing the purpose of the procedure and prompting you to confirm that you wish to continue.
2. Enter **y** at the confirmation prompt. You are asked to select the resilience option that you want to configure, from the menu:

```
Configure database for :

1. Transaction Handling
2. Stand-Alone Transaction Logging
3. Shadow
4. FailSafe/Heartbeat (Heartbeat on UNIX Only)
5. Stand-Alone Rapid Recovery

Enter option (1-5) :
```

## 3. Either:

- Select option 2 to configure both Rapid Recovery and Transaction Logging. When prompted:

```
Do you wish to set up database for Rapid Recovery File System?  
(y/n/q) ?
```

Enter **y** and follow the procedure for [Defining/Redefining a Transaction Logging database](#) described in the *Reality Resilience Manual*.

Or:

- Select option 5 to configure Rapid Recovery only. A message is displayed indicating that this will also enable Transaction Handling; enter **y** to confirm that you want to continue. Rapid Recovery is enabled on the database and you are then returned to the Configuration and Setup menu.

**Note:** Transaction Handling must be running in order for Rapid Recovery to be possible. If logging is stopped on the database – either by executing **TL-STOP** at TCL or via option 3 on **tlmenu**'s [Configuration and Setup Menu](#) – the Rapid Recovery feature is inhibited. It will not be possible to recover the database following a system failure until Transaction Handling is restarted.

## Recovery Procedure

When Reality restarts following a system or application failure, any Rapid Recovery database that has become corrupted is marked as 'Waiting for Recovery'. If a user tries to log on to one of these databases, the message `Awaiting Rapid Recovery` is displayed and the logon will fail.

Once you are satisfied that the underlying operating system is stable, execute the command

### **realrecover**

to initiate recovery of the database(s).

**Notes:** On UNIX you must be logged on as root to run **realrecover**.

The Rapid Recovery process is not repeatable. If a second system crash occurs while recovery is in process, the database will not be recoverable. This leads to problems if, for example, the system enters a power cycling mode. It is therefore recommended that you do not execute the `realrecover` command until you are confident that the system is stable. However, if you would like database recovery to be automatic when the system restarts after a failure, you can achieve this by adding the line `REALRAPIDRECOVERY=1` into the file `$REALROOT/files/realityrc`.

There is no need to restore a database from a backup tape. The Rapid Recovery Process will restore the database to a consistent state within minutes.

If a user tries to log on to a database while the recovery is in progress, the message `Rapid Recovery in Progress` is displayed and the logon will fail. When recovery is complete, the database is unlocked and users can log on.

When the database is restored to a valid structure, application updates made during the five minutes before failure are lost. For files that are Recoverable or Logged, logged updates are automatically replayed, but there will be application data missing from files that are Not Logged.

If the recovery process fails, the corrupted database remains locked. If anyone attempts to log on to the database, the message `This database needs checking` is displayed and the logon fails. It will then be necessary to restore the database from the most recent backup tape and, if the database is configured for Transaction Logging, to restore

updates from the clean log. Refer to [Transaction Logging Database Recovery](#) in the *Reality Resilience Manual* for a full description of this manual recovery procedure.

## Shadow Databases

If you have any shadow databases configured, the **realrecover** script will attempt to check and mount the base file system for each of these databases. Various messages are displayed as this process takes place.

If the mount of the file systems fails, the following message is displayed:

```
File system <mount_point> needs to be checked.
Please check and mount the above filing systems and then rerun this
script.
Alternatively rerun this script with the -f switch and the above
databases will not be recovered.
```

If you are unable to recover a shadow database via **realrecover**, you must follow the procedure described in [Shadow Database Recovery](#) in the *Reality Resilience Manual*.

## Actions Following Rapid Recovery

If any of the operations listed in the following table were in progress at the time of the system failure, you should take the appropriate action once the Rapid Recovery process has completed.

Operation in Progress	Action Following Rapid Recovery
Resize file	Restart the resize operation using <b>RESIZE-FILE filename (R)</b>
Create index	Delete the index and recreate it.
Account restore	Delete any incomplete accounts and restore them again.  <b>Note:</b> Account restore of a complete database is normally carried out before the database is enabled for logging. In this case the database cannot be recovered using Rapid Recovery: you should remake the database and restart the account restore.
MOVE-FILE	Check that there is no duplicate D-pointer. If there is a D-pointer in both the old location and the new location, use <b>EDELETE</b> to delete the old D-pointer.

---

# **Chapter 7**

## **Compressed Tape Image**

This chapter describes how you can specify a compression level for data saved in a tape image.

## Tape Images

A tape image is an ordinary file that is used to emulate a Reality tape device (on UNIX it can also be a named pipe). This provides a method of saving Reality data to disk. Any file name may be used, but it is recommended that the `.rti` extension is used so that tape images can be easily identified. For compressed images (see below), you could use the extension `.rci`.

Reality tape image emulates tape operations as closely as possible. When accessing a normal file (not a pipe), all normal tape operations are supported (T-FWD, T-REW, T-WEOF, etc.). When using a pipe as a tape image, you can only move forwards through the image; commands that rewind the tape (such as T-BCK and T-REW) are not available.

### Notes:

1. When accessing a tape image, a control file tracks the current position in the tape image. If you attach to a tape image, read a file, log off, log back on and re-attach to the tape image, you will be where you left off; that is, one file down the tape.

Always treat a tape image exactly as you would a real tape. If in doubt, do a T-REW before doing anything else.

2. As with tape drives, a tape image can contain multiple tape files.
3. Writing to a tape image will always write a full block, as it would to a tape drive. To save space in the image file use the compression feature described below.

## Data Compression

If required, you can specify that the data saved in the tape image should be compressed. This can be done in three ways:

1. Set the database configuration parameter `CompressTapeImage` to the compression level required. See Chapters [9 \(UNIX\)](#) and [14 \(Windows\)](#) of the Reality Reference Manual, Volume 3: Administration for details.
2. Set the operating system environment variable `REALCOMPTAPEIMAGE` to the compression level required. This overrides any `CompressTapeImage` setting.
3. Modify the path to the tape image to specify the compression level. This can be done in the [database configuration file](#), or by using the `T-DEVICE` command at TCL.

Specifying the level in this way overrides any default set with the previous two methods.

To specify the tape image compression, append `:clevel` to the tape image filename when defining the tape device, where *level* is the required compression level. For example, `/tmp/filsave.rti:c8` sets the compression level to 8 for the file `/tmp/filsave.rti`.

In all cases, the compression level must be a number from 0 to 9, where 0 is no compression (fastest) and 9 is maximum compression (slowest). The recommended compression level, optimising compression and performance, is 6. Compression is performed at the tape block level, so when using compression always use the maximum tape block size allowed to maximise compression.

The default is no compression, for compatibility with older versions of Reality. Note, however, that a compressed tape image cannot be read by versions of Reality earlier than V9.1. Reality V9.1 and later can read any tape image, whatever the compression level.

---

# Chapter 8

## Support for Distributed Transactions under MTS/COM+

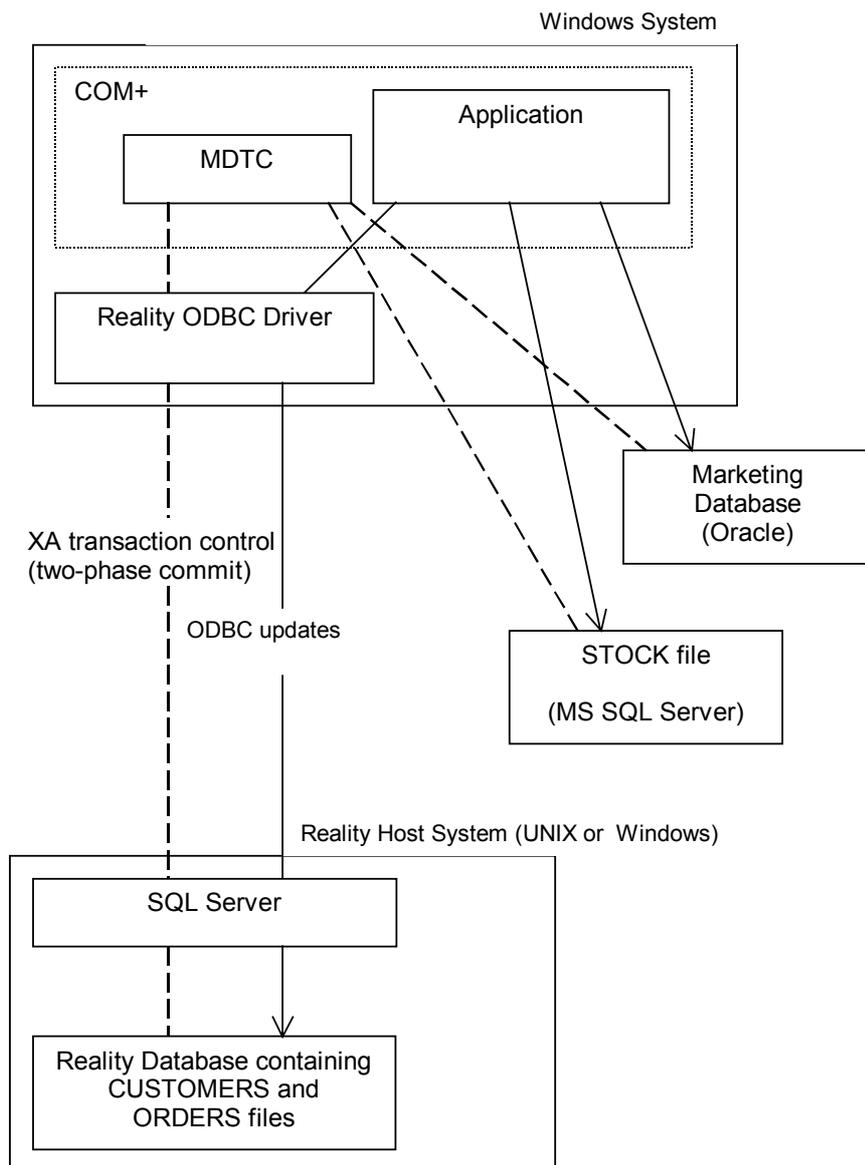
If an application accessing Reality via the SQL/ODBC interface is running in a Microsoft MTS/COM+ environment, it may be using *distributed transactions*. These transactions are fully supported by Reality, using both ODBC and XA interfaces.

## Distributed Transactions

SQL for Reality enables ODBC compliant applications to read and write data on a Reality database. If the ODBC compliant application accessing the database via the SQL/ODBC interface is running in a Microsoft COM+ environment, it may be using distributed transactions. Reality's support for COM+ transactions uses both ODBC and XA interfaces, see [Figure 8-1](#).

A distributed transaction, like a local transaction, is a set of related updates logically grouped by transaction boundary commands. However, the related updates can be to different databases on different systems on the network. If one of the updates is applied, all of the other systems must be updated to maintain consistency. In this scenario, updates applied to each database are classified as belonging to a transaction branch, each branch being uniquely identified by the Transaction Manager (TM). The responsibility for managing distributed transactions rests not with the local Reality TM, but with a remote TM: Microsoft's Distributed Transaction Co-ordinator (MDTC).

Where distributed transactions are being used, the Reality database must be configured for Transaction Handling and for Transaction Logging. Refer to [Defining and Starting/Stopping Transaction Handling](#) and [Configuring for Transaction Logging](#) in the *Reality Resilience Manual*.



**Figure 8-1. COM+ Distributed Transactions**

A local transaction is committed when the final update in that logical group is applied to the database. Up to that point, the transaction can be rolled back if remaining updates

fail to complete because of process or system failure, or because the transaction is deliberately aborted. The transaction is also rolled back if remaining updates fail to complete before the specified timeout period.

A distributed transaction has a two phase commit. In the first phase, each participating database is requested to **prepare** their transactions for commitment; then, providing they all return a successful outcome to phase-one, they enter phase-two whereby they are requested to **commit** their transactions. If one or more databases fail to prepare their transaction branches, the TM requests all databases to roll back their updates, effectively aborting the transaction. Once a transaction branch is prepared, the database effectively guarantees that it is capable of committing the transaction, even after a database crash. If there is a crash, the database may be requested to commit the transaction twice - once during normal operation and once after system recovery.

Until a transaction has been prepared, it can be rolled back, for example, if it exceeds the local transaction timeout value. The Reality SQL server informs the MDTC of the rollback and the MDTC rolls back all other participating transaction branch updates. Once a transaction is prepared, it cannot be rolled back by the local TM. In the case of a failure, the MDTC runs a recovery process, examining each prepared transaction branch in turn and requesting either a commit or a rollback

## MDTC Recovery Process

Recovery of processes using distributed transactions is handled by the MDTC.

In a distributed environment, failure can occur on the client, on the Reality server, or at any point on the network. Failures on the Reality server are of two types: process failure and system failure.

When a process fails:

- if the transaction branch has not been prepared, updates are rolled back, any locks taken are released, and the remote TM aborts the transaction.
- if the transaction branch has been prepared, it persists in that state - as do any locks taken on its behalf - even though the process itself may have terminated. When the MDTC initiates its automated recovery, it requests the transaction branch to either commit or roll back its transaction updates, effectively completing the transaction, and releasing all associated locks.

When a system failure occurs, Reality runs its standard transaction recovery phase, in which it scans the raw log and rolls back all uncompleted local transaction updates. During this period all Reality databases on the system are locked to all users. If the raw log contains any prepared distributed transactions, it is the responsibility of the MDTC to initiate and complete recovery of these outstanding transaction branches. When both local and distributed transactions have been resolved, the databases are released back to users.

If the MDTC itself fails, any outstanding prepared transaction branches will persist on the raw log until the MDTC restarts and connects to run the recovery process. This is known as cold recovery.

If the MDTC does not fail, it will connect to the Reality system immediately and start the recovery process. This is known as hot recovery.

When the MDTC connects to the Reality system to initiate the recovery, the connection is to the RXA server instead of to the SQL server as shown in [Figure 8-1](#). The RXA server provides the MDTC with details of all prepared transactions currently in the raw log. For each of these transactions, the MDTC issues a request to the RXA server to roll back or commit. When all of the rollback or commit operations have completed successfully, the MDTC closes the connection.

If the MDTC's automatic recovery process fails to complete, it is possible to carry out manual recovery of distributed transactions via the **rxaserver** command.

## **rxaserver Command**

f, following a system failure, the raw log contains prepared transactions for which the MDTC has not issued a request to rollback or commit; Reality databases on the system (all of which need access to the raw log) will remain locked to users. If the MDTC's automatic recovery process fails to complete, the **rxaserver** command allows you to manually commit or rollback prepared transactions.

---

### **Caution**

In order to make the decision to rollback or commit a prepared transaction, you must have information about the state of all other systems participating in that transaction.

---

At the UNIX or Windows system prompt, execute the following command to list all distributed transactions in the raw log:

#### **rxaserver -l**

```
Transaction 001 of 002
=====
Database : d:\databases\live DbKey : 1      Port : 401      Pid : 1896
Status   : ACTIVE           Started: 11:33 Offset: 0x08a18a00 Flags: 0x02
XID      : XA Transaction Identifier
FormatID : 0x4478019        Gtrid  : 48      Bqual  : 16
Data     : 9FB37754665F574D819A550FC9CAF9602364B472884F3F4EBA08302B03D699B1
          990B2E20B08A9C479BDABA08BC1C1FBD0B8F02E2CC317D41B1B6F4E260D0FDCD

Transaction 002 of 002
=====
Database : d:\databases\live DbKey : 1      Port : 404      Pid : 2073
Status   : PREPARED        Started: 11:35 Offset: 0x08a1c800 Flags: 0x82
XID      : XA Transaction Identifier
FormatID : 0x4478019        Gtrid  : 48      Bqual  : 16
Data     : F6737754665F574D819A550FC9CAF5102364B472884F3F4EBA08302B03D699B1
          990B2E20B08A9C479BDABA08BC1C1FBD0B8F02E2CC317D6AB109C2AAF020DE79
```

Execute the following command to run the manual recovery process:

#### **rxaserver -r**

The recovery process displays each prepared transaction in turn, giving you the option to abort or commit.

Transaction 001 of 001

=====

Database : d:\databases\live DbKey : 1 Port : 404 Pid : 2073

Status : PREPARED Started: 11:35 Offset: 0x08a1C800 Flags: 0x82

XID : XA Transaction Identifier

FormatID : 0x4478019 Gtrid : 48 Bqual : 16

Data : F6737754665F574D819A550FC9CAF5102364B472884F3F4EBA08302B03D699B1  
990B2E20B08A9C479BDABA08BC1C1FBD0B8F02E2CC317D6AB109C2AAF020DE79

Enter 'I'gnore (default), 'A'abort, 'C'ommit : **C**

You are about to Commit transaction 1 do you want to continue? <Y/N> : **Y**

Operation to Commit transaction 1 was successful, result 0

---

# Index

@ debugger command 5-42

## A

ACCESS function 5-2, 5-35

## B

Breakpoints, inhibiting debugger breaks  
5-42

## C

CLEARFILE statement 5-2  
COM+ 8-2  
    transaction example 8-3  
CompressTapeImage 7-2  
Configuration parameters,  
    CompressTapeImage 7-2  
Conversion codes 4-8  
CREATE-TRIGGER command 5-25

## D

Data compression, tape image 7-2  
DEBUG statement 5-42  
Debugger prompt 5-5  
Definition items, file 4-6  
DELETE-TRIGGER command 5-26  
Deleting file triggers 5-26  
Directory view 4-2

Distributed transaction 8-2

    example 8-3  
    manual recovery 8-6  
    recovery 8-5

## E

Environment variables,  
    REALCOMPTAPEIMAGE 7-2

## F

FDB-CLEAR command 4-9  
FDB-SET command 4-10  
FDB-SHOW command 4-11  
FDISCTOTAPE command 5-26  
File definition item 4-6  
File triggers 5-2, 5-25, 5-26, 5-28, 5-35  
File types 4-2  
Foreign databases 4-12  
    creating files 4-10, 4-11  
    Reality files on foreign databases 4-7  
    saving and restoring files 4-11  
    SQL-view files 4-12

## I

Inhibiting debugger breaks 5-42  
INPUTERROR statement, in triggers 5-2

**L**

LISTFILES command 4-3  
Listing file triggers 5-28  
LIST-TRIGGERS command 5-28  
Locks  
    retrieval 4-8  
    update 4-8  
Logging status 6-2

**M**

MDTC 8-2  
Microsoft Distributed Transaction Co-ordinator 8-2  
Modulo 4-9  
MultiValue systems  
    transferring data from 5-26  
    transferring data to 5-30

**O**

ODBC  
    Data Source 4-10, 4-12  
    interface 8-2  
    Reality files on foreign databases 4-8

**P**

Pipe, using as a tape image (UNIX) 7-2  
Process recovery, distributed transactions 8-5  
Pseudo-floppy image  
    converting to Reality tape image 5-26  
    creating from tape image 5-30  
Pseudo-tape – see Tape image

**Q**

Q-pointer 4-3, 4-4  
Quick images 6-2

**R**

Rapid Recovery  
    database recovery 6-6  
    description 6-2  
    shadow databases 6-7  
Raw log 6-2  
REALCOMPTAPEIMAGE environment  
    variable 7-2  
Reallocation parameters 4-9  
**realrecover** command 6-6  
Recoverable Files 6-2  
Retrieval locks 4-8

**S**

Shadow database, Rapid Recovery 6-7  
SQL-VIEW command 4-12  
SQL-view files 4-2, 4-12  
SYSTEM statement 5-41

**T**

Tape image 7-2  
    converting to pseudo-floppy image 5-30  
    creating from pseudo-floppy image 5-26  
    data compression 7-2  
TAPETOFDISC command 5-30  
Transaction, distributed 8-2  
Transferring data  
    from other MultiValue systems 5-26  
    to other MultiValue systems 5-30  
Triggers – see File triggers

**U**

Update locks 4-8

**W**

WF debugger command 5-43  
WS debugger command 5-44

**X**

XA interface 8-2