

AutoTestSQL

Operation Manual

AutoTestSQL

Operation Manual

by Liam Elliott

This manual is also available as 'on-line help' from the AutoTest software. You can access the on-line help from the 'Help' menu. The on-line version is context-sensitive: by pressing F1, you can get immediate help for whichever menu or dialogue box you are currently using.

Table of Contents

Part 1	General information	1
Part 2	Introduction to AutoTestSQL	3
1	About this manual	3
Part 3	Operation overview	7
1	How it works	7
2	Getting started	7
3	User interface basics	9
4	AutoTestSQL modes	10
5	Reporting	10
6	AutoTestSQL and dScope	10
7	Licensing	11
8	Information about this software release	11
Part 4	Test structure	15
1	Test structure overview	15
2	Test Sets and Test Elements	16
3	Result Sets and Result Elements	17
4	How a result status is calculated	18
5	Scriptlets and their responsibility	19
Part 5	Test Scriptlets	23
1	Introduction to VBScript	23
2	Product and Test Set Scriptlets	24
3	The User Script	24
4	The setResult function	25
5	Considering Fault-find mode	26
6	Script functions	28
	Optional Scriptlet functions	28
	OnTestStart.....	28
	OnTestFinish.....	29
	ValidateSerialNumber.....	30
	Built-in functions	31
	SetResult.....	31
	GetLimits.....	32
	GetMode.....	33
	GetProductFamily.....	34
	GetProduct.....	34
	GetSerialNumber.....	34
	GetOperatorName.....	35
	GetOperatorID.....	35

	TestStopped	35
	TestFailed.....	36
	StopTest.....	36
	Functions in the User Script	36
	ColourMsgBox	37
	FailMsgBox.....	39
	Colours.....	42
	Alignment.....	42
7	Writing and debugging scripts	43
	Choosing a script editor	43
	Writing scripts	44
	Debugging scripts	45
	Common script problems	46
	Reusability of scripts	47
Part 6	Operation reference	51
1	Menus	51
2	Toolbar	52
3	Setup mode	53
	Setting up Products	54
	Entering/editing Product Family details.....	55
	Entering/editing Product details.....	55
	Setting up tests	56
	Entering/editing Test Set details.....	58
	Entering/editing Test Element details.....	59
	Setting up Operators	60
	Entering/editing Operator Details.....	61
4	Test mode	62
	Fields and controls	63
	Tree of results	65
	Running a test	66
	Stopping a test	67
	Re-testing failures	67
	Test completion	68
	The Test Report	70
5	Fault-find mode	71
	How Fault-find mode works	71
	Fields and controls	72
	Selecting a failed result	74
	Entering fault details	76
	Knowledge of the test structure	77
6	Reports	78
7	Options dialogue box	79
8	AutoTestSQL security	82
Part 7	The AutoTestSQL database	87
1	Database structure	87
2	Database tables	88
3	Using SQL from the command-line	93
4	Changing the database administrator password	93
5	Creating new database logins	94

6	Backing up and restoring the database	94
7	Multi-user setup	95
8	Limitations of MSDE	96
Part 8	Management reporting	99
1	Connecting to the database	99
2	The AutoTestSQL Report Viewer	100
	User interface basics	100
	Menus	102
	Options dialogue box	102
	The Crystal Report viewer	103
	Creating your own report templates	104
Part 9	Glossary	109
	Index	114

Part



1

General information

1 General information

Manual revision history

Rev	Date	Author	Notes
0.94	26th April 2004	L.R.Elliott	To accompany first beta software release (V0.94).
0.95	7th October 2004	L.R.Elliott	To accompany Beta release V0.95
0.96	4th February 2005	L.R.Elliott	To accompany release V1.00

Support contacts

Prism Media Products Limited	Prism Media Products Inc
William James House	21 Pine Street
Cowley Road	Rockaway
Cambridge CB4 0WX	NJ 07866
UK	USA
Telephone: +44 1223 424988	Telephone: +1 973 983 9577
Fax: +44 1223 425023	Fax: +1 973 983 9588

Email: tech.support@prismsound.com

Web: <http://www.prismsound.com>

Or contact your local Prism Sound distributor as detailed on the website.

Trademark acknowledgements

Access, ActiveX, Microsoft, MSDE, SQL Server, Visual Basic, VB, VBA, VBScript and Windows are trademarks of Microsoft Corporation.

Crystal Reports is a trademark of Business Objects.

MySQL is a trademark of MySQL AB.

All trademarks acknowledged.

© 2004-2005 Prism Media Products Limited. All rights reserved.

This manual may not be reproduced in whole or part, in any medium, without the written permission of Prism Media Products Limited.

In accordance with our policy of continual development, features and specifications are subject to change without notice.

Part



2

Introduction to AutoTestSQL

2 Introduction to AutoTestSQL

AutoTestSQL is an automated test management system that allows a series of tests to be performed on a number of different Products. It is designed to work in conjunction with the dScope Series III Audio Test & Measurement system. Tests are specified in a hierarchical structure, and stored in a [database](#). Small [Scriptlets](#) (written in the [VBScript](#) programming language) are written to perform individual parts of the test; when the test is run, these write results into the database. The database can be later queried for results, and any number of reports created for the details stored.

For a quick overview of AutoTestSQL operation go to [Operation overview](#).

For an in-depth description of setting up a test structure, see [Test structure](#).

For a guide to writing Test Scriptlets for AutoTestSQL, see [Test Scriptlets](#).

For an in-depth reference to using AutoTestSQL, see the [Operation reference](#).

If you would like to delve deeper into the way the AutoTestSQL database works, see [The AutoTestSQL database](#).

2.1 About this manual

The AutoTestSQL Operation Manual is provided in two different formats: as a conventional printed manual, and also as 'on-line help' which can be viewed whilst operating AutoTestSQL. The printed version is also provided in 'electronic' format, as a 'pdf' file, with the AutoTestSQL software. These files can be viewed and printed using the Adobe Acrobat Reader, which can be downloaded free at www.adobe.com. Updates of the manuals are available from the Prism Sound website at www.prismsound.com.

When viewed on-line, the manual pages are accompanied by a navigation area to the left. Therein, a "Contents" section shows a hierarchical map of the entire document from which desired pages can be selected. Next to "Contents", the "Index" section allows instant access to pages describing particular topics. The "Search" section lists all pages containing a particular word or phrase, and the "Favorites" section can be used to save page locations for future reference.

Entry into the on-line help from the AutoTestSQL application is 'context-sensitive', so pressing the F1 key takes you directly to the help page for whichever dialogue box or panel you are using at that time.

When viewed as on-line help, each page is headed by a title block which shows the name of the page, plus some links on the right-hand side. The upper row of links refer to topics above the current page in the manual's hierarchy. Below, a "See Also" link often appears which accesses a pop-up box containing a list of related topics.

Within the body of each page, certain font and highlighting conventions are used:

Links to other parts of the manual are shown [like this](#).

Buttons on the AutoTestSQL dialogue boxes are designated, for example, [OK].

Code samples are shown in this font...



Noteworthy items are indicated like this.



Important warnings are designated like this.

Part



3

Operation overview

3 Operation overview

This section gives an overview of the operation of the AutoTestSQL application. For a more in-depth reference, see the [Operation reference](#) section.

For a brief guide to AutoTestSQL operation, see [How it works](#).

For some important information on getting started with AutoTestSQL, see [Getting started](#).

For a description of the AutoTestSQL user interface, see [User interface basics](#).

If you would like to see the changes in this version of the software, please see [Information about this software release](#).

3.1 How it works

AutoTestSQL uses a database to store details of tests and results for a range of different Products. It allows the Operator to enter and edit a series of tests, grouped in a hierarchical structure. These tests are associated with a number of [Scriptlets](#), written in the [VBScript](#) programming language, that perform the tests themselves (usually with the aid of the dScope software). The results are then written into the database; from here they can be used to produce full test results for an [EUT](#), or can be used for management reporting purposes.

If a failure occurs during a test, AutoTestSQL can be used to pin-point the failure, analyze the reasons for failure, and record its details in the database. This provides a way of analyzing the most common problems, and thereby reducing their occurrence.

AutoTestSQL has three different modes of operation that it uses to perform these tasks. **Setup mode** allows the Products and test structure to be entered, and **Test mode** allows the tests to be performed. **Fault-find mode** allows the system to be taken back to the state it was in at any point of failure, in order for that failure to be analyzed and fixed, and its details to be logged. For further details of these modes, see [AutoTestSQL modes](#) for an overview, or the [Operation reference](#) for an in-depth description.

AutoTestSQL reporting is currently external to the AutoTestSQL application. It is proposed that a professional Report Management tool (such as Crystal Reports) will be used for the most flexible and useful reporting capability. See [Management reporting](#) for further details.

3.2 Getting started

Before starting to use AutoTestSQL, please read the [Operation overview](#). This will give you a basic background. For full details of setting up tests and writing scripts, you will need to refer to the sections on [Test structure](#) and [Test Scriptlets](#).

[MSDE](#) is the [database](#) used by the AutoTestSQL software. When MSDE is first installed, the database itself will have a system administrator user with a blank password. To prevent unauthorized access to the AutoTestSQL database, it is important that this password is changed. See [Changing the database administrator password](#) for details.

It is also important to back up your database regularly. See [Backing up and restoring the database](#) for details of how to do this.



When running AutoTestSQL for the first time, you will be prompted to log in. Initially, there is only one Operator available on the system, the Administrator, so you will have to log in as this Operator. The default password for the Administrator is blank, but you will be asked to change this as soon as you have logged in for the first time.

Terminology

Some names of parts of AutoTestSQL can be changed. For example, you may wish to call a "Product Instance" an EUT, and an "Operator" a User. These changes can all be made using the [Options dialogue box](#), available from the Utility menu.

Within this manual however, the names of all these items are given the AutoTestSQL defaults. These are:

Product Family: A group of Products.

Product: The actual items that are tested.

Product Instance: Individual instances of Products. A Product Instance equates to a single item of Equipment Under Test (EUT).

Operator: A user of AutoTestSQL, that sets up the tests and performs testing and fault-finding.

Scriptlet: An individual script file that is associated with a Test Set.

Test Set: A group of tests. This group may contain Test Elements, and/or other Test Sets.

Test Element: A Test entity that corresponds to a single value (numerical, pass/fail or text) that will be entered into the database.

Result Set: A group of results. This group may contain Result Elements, or other Result Sets.

Result Element: A single result value that will be entered into the database.

3.3 User interface basics

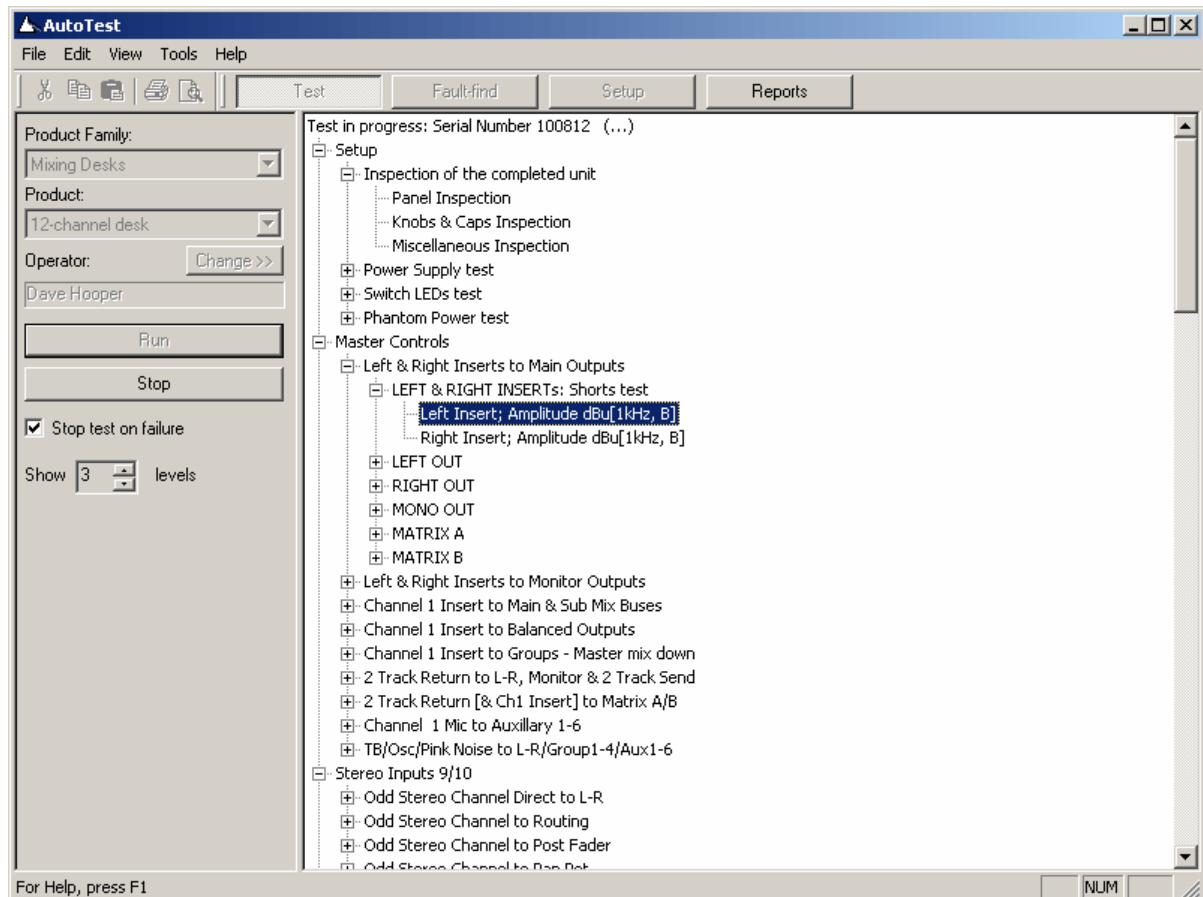
The user interface of AutoTestSQL comprises a number of basic elements:

[Menu bar](#)

[Toolbar](#)

[Main window](#) (two vertical panes)

[Status bar](#)



[Menu bar](#)

The Menu bar is situated at the top of the AutoTestSQL window. It provides access to several functions of AutoTestSQL that are not available on the main Toolbar.

[Toolbar](#)

Below the Menu bar is the Toolbar. This contains buttons for some common Windows tasks together with buttons used to access the main [modes](#) of AutoTestSQL.

[Main window](#)

The main part of the AutoTestSQL window is split vertically into two panes. The contents of each of these panes is dependent on the current mode of operation of AutoTestSQL.

Status bar

The bottom line of the AutoTestSQL window is the Status bar. This shows important indications of the current state of AutoTestSQL, including warning messages.

Full details of all these parts of the user interface can be found in the [Operation reference](#) chapter.

3.4 AutoTestSQL modes

AutoTestSQL can operate in three different modes. These modes are used for different stages of the test process.

This section gives a brief overview of each of the modes; for full details, see the [Operation reference](#).

[Setup mode](#) is used to initially define the Products that will be tested, and the tests that will be performed on them. It allows you to specify a hierarchical structure of tests for each Product, and associate the tests with [Scriptlets](#) that will be used to run the tests. It also allows setup of the different Operators that will use the system, and allows specification of the different tasks that each one can perform.

[Test mode](#) is used to run the tests. This is the most common mode of operation, and is the mode entered when AutoTestSQL is started. When a test has finished, all results can be logged to the database for future reference or analysis. Upon completion of a test, a Test Report can be produced if required.

[Fault-find mode](#) can be used after a test has failed, to find out more information about the failure. Once the point of failure has been selected, AutoTestSQL will take the Operator back to that point in the test. Other equipment and applications (for example, the dScope software, or a multi-meter) can then be used to determine the cause of the fault. If required, the details of the fault can be written to the database for later analysis.

3.5 Reporting

There are two types of report in AutoTestSQL:

A **Test Report** can be produced if required at the end of a test, giving pass/fail details of individual parts of the test. This is structured with the same hierarchy used to set up the tests. For full detail of this report, see the section on [The Test Report](#) in the Operation reference.

Management reports can be created by querying the database for information about a test or range of tests. For example, you could create a report that lists all the failures that occurred for a given product line; or you could list all the tests done within a time period and whether they passed or failed. For full details on management reporting, see [Management reporting](#).

3.6 AutoTestSQL and dScope

The Windows operating system allows different pieces of software to communicate with each other using a process known as [OLE](#) Automation. The dScope software uses this technology to allow itself to be automated by other applications. It does this by exposing an interface, defined in a standard language called "Object Definition Language", to the Windows operating system. This interface is defined in a [Type library](#), and any Windows program that can control automation-enabled objects can control the dScope.

The AutoTestSQL software is just such an application. AutoTestSQL uses [VBScript](#) to run its tests,

which has the capability of using any application with an automation interface. It can therefore control the dScope software, and use any of the exposed methods or properties of its interface.

AutoTestSQL creates an instance of the dScope application when Test mode or Fault-find mode is entered. This dScope application can then be used by Scriptlets throughout the test process. The dScope application is shut down again when the mode is no longer Test or Fault-find (for example, entry of Setup mode, or closing of the AutoTestSQL application).



When dScope is started, AutoTestSQL will check this dScope for a valid licence key. For further details, see the [Licensing](#) section of this manual.

3.7 Licensing

The AutoTestSQL software is licensed for use with a single dScope Series III. This means that Test mode or Fault-find mode can only be run if the appropriately licensed dScope is attached to the computer that the AutoTestSQL software is running on. Setup and Report modes will always be available, regardless of whether a valid licence is present.

When Test or Fault-find mode is entered, AutoTestSQL will attempt to start up a dScope attached to the system. If successful, this dScope's serial number will be read and checked against the AutoTestSQL licences on the PC. If no valid licence is found, then the Operator will be informed and AutoTestSQL will enter [Setup mode](#).



If you have multiple copies of AutoTestSQL, and several dScopes, then the licences for each dScope can be applied to each computer running AutoTestSQL, so that you can use the AutoTestSQL software with *any* of the licensed dScopes.

AutoTestSQL can be run for a limited period of 30 days from installation without a licence.

To obtain a licence for AutoTestSQL, please contact Prism Sound using the details in the [General Information](#) page.

3.8 Information about this software release

Although we have tried to make AutoTestSQL's operation as generic and useful as possible, there may be areas where you find that the functionality does not suit your purposes very well. We welcome your comments regarding the software's feature set, and any bugs in the software that may be found. These comments will be added to a list of future issues to be included in the software.

Please e-mail your comments, suggestions and bug reports to tech.support@prismsound.com.

[New features](#)

The following changes have been made to the software between V0.95 and V1.00

- Name changed to AutoTestSQL
- Software licensing added
- Ability to select which mode AutoTestSQL starts up in.

The following changes were made to the software between V0.94 and V0.95

- Ability to re-test the last Test Set, if a failure occurs during the test (See [Re-testing failures](#)).

- Database access no longer occurs throughout the test, to speed up the testing process. Results are now written to the database at the end of the test.
- Product Instance table has been removed from the database (See [Database structure](#) and [Database tables](#)).
- Option of whether to save untested results to the database, and how to mark items that have been retested. (See [Options dialogue box](#)).

Part

4

Test structure

4 Test structure

Before using AutoTestSQL to test your Products, you must define the tests that are to be performed. To do this, you will need to decide on a structure for these tests. Once this structure exists, tests can be easily created within the AutoTestSQL application.

For an overview of how tests are structured, see [Test structure overview](#).

For details of Test Sets and Test Elements, the constituent parts of the test structure, see [Test Sets and Test Elements](#).

For information on how the structure of results corresponds to this test structure, see [Result Sets and Result Elements](#).

To see where Scriptlets fit into this test structure, see [Scriptlets and their responsibility](#).

4.1 Test structure overview

Within AutoTestSQL, tests are structured hierarchically to enable sensible grouping of results.

An individual item in a test that will produce a single result (whether numerical, pass/fail, or text) is called a **Test Element**. Test Elements can be grouped together within **Test Sets**, which can be grouped in turn into further Test Sets, to a maximum of 10 levels.

For example, part of a test structure for a stereo product could be:

```
XLR Input Tests
  Channel 1
    Amplitude (dBu)
    THD+N (dB)
    Cross-talk (%)
  Channel 2
    Amplitude (dBu)
    THD+N (dB)
    Cross-talk (%)
```

In this simple example, XLR Inputs Tests, Channel 1, and Channel 2 are all **Test Sets**, while Amplitude, THD+N and Cross-talk are **Test Elements**. Each Test Element corresponds to a single result value that will be stored in the database. The Test Element is set up with limits; every result that is read for this Test Element will be compared to these limits when written to the database, to give the corresponding Result Element a status of pass or fail.

A Test Set grouping can be anything that is sensibly grouped in a list of test results. For example, you are likely to perform similar tests for each channel of a stereo or multi-channel product; therefore the group of results for each channel would make a sensible Test Set. The tests that you do on your [EUT](#) inputs could be grouped together in a Test Set; likewise, the tests done on the EUT Outputs could be another Test Set.

When a test is run, a set of results will be created in the database for the test. The way that the results are structured corresponds exactly to this test structure - i.e. A Result Set will be created for each Test Set in the test structure, and a Result Element will be created for each Test Element. This allows test results to be sorted in such a way as to mirror the structure used when setting up the tests.

For full details of Test Sets and Test Elements, see [Test Sets and Test Elements](#). For a description of Result Sets and Result Elements, see [Result Sets and Result Elements](#).

4.2 Test Sets and Test Elements

This section describes in detail the two constituent parts of a test structure. For an overview, see [Test structure overview](#). For full details on setting up Test Sets and Test Elements, see [Setting up tests](#) in the [Operation reference](#).

Test Sets

A Test Set is any grouping in the test structure of Test Elements, or other Test Sets. Test Sets exist for two purposes: Firstly, to group together Test Elements or other Test Sets in a list of test results; and secondly, to act as a placeholder for [Scriptlets](#) that actually perform the tests and record the results.

When setting up a Test Set, you define the following:

Name: The name of the Test Set.

Scriptlet: The file name of an optional [Scriptlet](#) associated with the tests within this Test Set. For full details of how these Scriptlets work within the test structure, see [Scriptlets and their responsibility](#).

For full details of entering or editing a Test Set in Setup mode, see [Entering/editing Test Set details](#).

Test Elements

A Test Element is part of the test structure at the lowest level that corresponds to a *single result value* that will be stored in the database. For example, a single measurement of THD+N to be made on channel A will be entered into the test structure as a Test Element. If a [sweep](#) is to be done (for example a frequency response), then each point in the sweep will be a different Test Element, since each point will correspond to a separate value in the test results.

Test Elements have a number of different properties that can be defined:

Description: This describes exactly what the result value means. For numerical values, it is important that this description (or the description of its parent Test Set) contains details of the unit in which the value is expressed. For example, a Test Element representing an Amplitude measurement should specify in the description the unit in which the amplitude is measured in, e.g. "Amplitude (Vrms)". For a series of measurements, this information could be specified in the parent Test Set - e.g. for a frequency response sweep, the Test Set could be called "Frequency response (dB)", with each Test Element's description simply detailing the frequency at which the value was measured. In the amplitude example, the test results would list the Result Element as:

Amplitude (dBu) : 12.43

In the frequency response example, the results would be shown in the following way:

Frequency response (dB)
20Hz: -0.30
100Hz: -0.01
1kHz: 0.00
5kHz: 0.02
20kHz: -0.04

The structuring of this description is entirely up to you, but you can see that without specifying the unit in the description of the Test Element or the name of the Test Set, the values would be meaningless to anyone looking at the test results.

Result type: This defines the type of result that will be written into the database. It can be one of three types: a numerical value, a pass/fail value, or a text string. The type of result, together with the Upper and Lower limits, will determine how the status of the result is stored. See [How a result status is calculated](#) for full details.

Use ... decimal places when displaying results: This applies only to numerical values and is used

when displaying a Test Report. It simply determines how the numbers are rounded for display purposes.

NB: This property defines *only* how the result is displayed on test results and test reports; The number itself is stored in the database as a [double-precision](#) number to the same accuracy as was used by the [SetResult](#) function call in the test [Scriptlet](#).

Upper & Lower Limits: For numerical and text string values, these limits define how the status of the corresponding Result Element will be calculated. See [How a result status is calculated](#) for full details.

For full details of entering or editing a Test Element in Setup mode, see [Entering/editing Test Element details](#).

4.3 Result Sets and Result Elements

When a test is performed, a [Test Session](#) record is entered in the [database](#). This contains details of the date and time of the test, together with a link to the current [Operator](#) and the [Product Instance](#) that is being tested. Under this Test Session, a set of results is created that corresponds to the test structure. This set of results is structured as Result Sets and Result Elements.

[Result Sets](#)

A Result Set is created to represent the Test Set in the set of results. It is linked to its parent Result Set (if it is not at the top level), and also to the Test Set from which it has been created, giving access to the name of the Test Set. It exists solely as a way of grouping together Result Elements and other Result Sets.

[Result Elements](#)

A Result Element is created to store a single result value in the database. It is linked to a parent Result Set, and also to the Test Element from which it was created. This gives it access to the Test Element's description, together with the limits and result type.

Since Result Sets and Result Elements are stored in the database in an identical structure to the Test Sets and Test Elements, it is easy to extract results from the database in the hierarchical order in which the test structure was created. Even if the test structure is subsequently changed, existing sets of results will retain the original Test Set structure which was mirrored when the results were created.

4.4 How a result status is calculated

When a Result Element is written to the database using the [SetResult](#) function, its result status is calculated for the purposes of displaying the status on a Test Report, and giving an indication on the result display whether a test, or part of a test, has failed.

A result status can be one of four possible values:

Passed	This part of the test has passed. This status is usually represented on the display or on Test Reports with the word "Passed", in green.
Failed	This part of the test has failed. This status is usually represented on the display or on Test Reports with the word "FAILED", in red.
Unknown	AutoTestSQL cannot work out whether this result has passed or failed. This is common when a test is run, because AutoTestSQL must create Result Elements in preparation for writing results to the database. However, until the relevant Scriptlet has called SetResult , the status is unknown. This status is usually represented on the display or on Test Reports as "...".
Non-critical	This status applies to a result for which no limits have been set. Since there are no limits, the result is assumed to be for information only. Non-critical results do not have any associated status on the display or on Test Reports.

[Result types and Limits](#)

Each Test Element (and therefore each corresponding Result Element) can have one of the following result types. For each one, the result status is calculated differently according to the upper and lower limits specified:

Numerical value	<p>If neither an upper nor lower limit has been specified, then the result status will be Non-critical.</p> <p>If an upper limit has been specified, and the result value is above this limit, then the result status will be Failed.</p> <p>If a lower limit has been specified, and the result value is below this limit, then the result status will be Failed.</p> <p>Otherwise, the result status will be Passed.</p>
Pass/fail	<p>The SetResult function can accept either a boolean variable (True or False) as the result parameter, in which case the result status will be Passed or Failed, respectively.</p> <p>Alternatively, a string can be passed to the SetResult function, in which case AutoTestSQL will try to determine whether the string represents a pass or a failure. AutoTestSQL will treat values of "Valid", "True", "Yes" or anything beginning with "Pass" as Passed; "Invalid", "False", "No" or anything starting with "Fail" will be flagged as Failed. Anything else is treated as Unknown.</p> <p>Note that for pass/fail result types, the upper and lower limits are irrelevant for this type of result, as the status is implicit in the result value.</p>
String	<p>If neither an upper nor a lower limit have been specified, the result status will be Non-critical.</p> <p>If an upper limit only has been specified, then the result string must match this string to be stored as Passed. Otherwise it will be Failed.</p> <p>Similarly, if a lower limit only has been specified, then the result string must match this string to be stored as Passed. Otherwise it will be failed.</p> <p>If both the upper and lower limits have been specified, then the result string must match one of these to be stored as Passed; otherwise, it will be Failed.</p>

[Status of parent Result Sets and Test Sessions](#)

When a Result Element's status is calculated, this will affect the status of its immediate parent in the result structure, and all the parent items recursively up to the Test Session at the top of the set of results.

If any Result Element fails, then its parent Result Set is also flagged as failed; moving recursively up the tree of results, each parent Result Set will be flagged as failed and therefore the Test Session as a whole is a failure.

If a Result Set has *some* Result Elements that have passed, and the rest are unknown (because results have not yet been stored), then the parent Result Set's status is unknown (it cannot be known until the status of all the Result Elements have been set). Thus, the status of the Test Session as a whole is unknown, unless another Result Set within it has failed, in which case the Test Session is also a failure.

The only way a Result Set can pass is if *all* its Result Elements and child Result Sets have themselves passed (or are non-critical). In the same way, a Test Session can only pass if *all* its Result Sets have passed or are non-critical.

If *any* Result Element in a set of results is a failure, then the Test Session as a whole is marked as a failure.

4.5 Scriptlets and their responsibility

A test structure is grouped into Test Sets, each of which can have any number of Test Elements and/or child Test Sets. Each Test Set can have a [Scriptlet](#) associated with it, that can perform part, or all, of the tests required for that Test Set.

This section gives an overview of how Scriptlets fit into the test structure. For a full reference, see the [Test Scriptlets](#) section.

There are two places where a Scriptlet can be defined: For the Product itself, or for each Test Set. These Scriptlets have slightly different responsibilities:

[Product Scriptlet](#)

A Product Scriptlet can contains [constants](#), [variables](#), [functions](#) and [subroutines](#) that can be used throughout the tests for this Product.

[Test Set Scriptlet](#)

A Test Set Scriptlet contains code for an individual test or group of tests. It can run the tests (and store the results) for *any* of the Test Elements below it in the test structure.

It is important to consider carefully the location of a Test Set Scriptlet in the test structure, and limit the number of tests performed by a single Scriptlet, because of the way [Fault-find mode](#) works. An example of the options available is given below; to find out more about how Fault-find mode uses the Scriptlets in the test structure, see [Considering Fault-find mode](#).

[Example](#)

For example, consider the following part of a test structure. Note that in reality a test structure would be much more complicated than this, and the issues raised will therefore apply on a larger scale:

```
D/A Converter
  Audio Tests
    XLR Input Tests
      Channel 1
        Amplitude (dBu)
        THD+N (dB)
        Cross-talk (%)
      Channel 2
        Amplitude (dBu)
        THD+N (dB)
        Cross-talk (%)
    XLR Output Tests
  ..
Other Tests
..
```

In this case, the top level is the Product, the next three levels are Test Sets and the lowest level represents the Test Elements. Within each channel shown here, three measurements need to be taken (Amplitude, THD+N, and Cross-talk). It must be the responsibility of a Scriptlet in one of the Test Sets above these to make the measurements and record them in the database.

There are three possibilities here:

[Option 1](#)

The "Channel 1" and "Channel 2" Test Sets could *each* have an associated Scriptlet that sets the channel, then makes the three measurements. This could even be the same Scriptlet, providing that the channel being measured was set before taking the measurements.

The "XLR Input Tests" and "Audio Tests" Test Sets then have no responsibility to make any measurements, and in fact do not need to contain Scriptlets at all, since the Test Sets below it are making all the necessary measurements. However, they could be used to change any relevant settings (such as loading a dScope [Configuration](#)) that were relevant to the tests below them.

Option 2

The "XLR Input Tests" Test Set could have a script that makes *all* the measurements below it in the test structure - i.e. Amplitude, THD+N and Cross-talk on two channels. This would mean that the "Channel 1" and "Channel 2" Test Sets would not contain Scriptlets, since there is nothing for them to do. The "Audio Tests" Test Set does not need to make any measurements, although as in the example above, it could be used to change any relevant settings before the measurements are made.

Option 3

The "Audio Tests" Test Set could contain a Scriptlet that makes all the measurements for all the Test Elements in Test Sets below it. This leaves "XLR Input Tests", "Channel 1" and "Channel 2" with nothing further to do. However, it means that the Scriptlet would be a considerable size since it has to make ALL measurements in "XLR Input Tests" and "XLR Output Tests".

Of these choices, option 2 is the best approach. It can make use of the dScope's ability to make measurements on both channel A and channel B at the same time, thus speeding up the tests. Option 1 does not make use of this ability, and option 3 will result in an enormous Scriptlet that will cause problems in fault-find mode (since fault-find mode would have to run the entire Scriptlet up to any point of failure within it).

Part



5

Test Scriptlets

5 Test Scriptlets

Within AutoTestSQL, [Scriptlets](#) can be written to manipulate the dScope and the [EUT](#), make measurements, and record results in the database. Different Scriptlets are set up to perform different parts of a test; these are run in the order defined by the test structure to perform the complete test.

Scriptlets are written in the [VBScript](#) programming language. For more details, see [Introduction to VBScript](#).

For a description of the difference between Product Scriptlets and Test Set Scriptlets, see [Product and Test Set Scriptlets](#).

For an in-depth reference of built-in functions and subroutines, see [Script Functions](#).

5.1 Introduction to VBScript

[VBScript](#), or Microsoft Visual Basic Scripting Edition, is a lightweight programming language. It was originally designed to help to automate web pages, and has evolved to become a useful [automation](#) tool. It is a subset of Visual Basic, and contains many of the same language features.

VBScript allows use of simple programming constructs - for example, loops and conditional statements. It can also use ActiveX controls which greatly increases its functionality (see [VBScript and ActiveX Controls](#), below) and this allows AutoTestSQL to control other software. This enables it to work in conjunction with the dScope, since the dScope software has a built-in automation interface allowing access to all its settings via properties and methods.

It is beyond the scope of this Help file to provide a detailed description of VBScript; there are a number of resources available to help you out in this respect. [Documentation on Windows Script Technologies](#), including VBScript, can be downloaded from the Microsoft web site to give a detailed description of VBScript functionality.

[VBScript and ActiveX Controls](#)

One of the main benefits of VBScript is that it can create and use [ActiveX](#) controls. This enables it to create instances of other software applications, and use any properties and methods exposed by such software. This is the way that AutoTestSQL works in conjunction with the dScope. It also means that other useful pieces of test equipment can be used, if they have an ActiveX interface - for example, digital multi-meter PC cards.

Creation of an ActiveX control from a script is very simple. You simply have to create an object of the relevant type; the objects and properties of this method can then be used in the script by prefixing with the name of this object.

As an example, consider creation of the dScope object from VBScript:

```
' Variables
Dim dScope      ' The dScope object

' Create the dScope object
Set dScope = CreateObject("dScope.Application")
```

Any of the dScope's properties and methods can then be used by using them with a "dScope." prefix.

See the descriptions of [OnTestStart](#) and [OnTestFinish](#) for a full example of using the dScope within a Scriptlet.

The Scriptlets can also use the [ScriptDlg ActiveX Control](#) provided with the dScope to allow specific user interface options to be shown to the Operator - for example, drop-lists, buttons and slider controls. Please see the section on the ScriptDlg ActiveX Control in the dScope Scripting Manual for a full reference, or the sections on [ColourMsgBox](#) and [FailMsgBox](#) for some examples of how it can be used.

5.2 Product and Test Set Scriptlets

When setting up a test structure, [Scriptlets](#) can be associated with the Product itself, and for any or all of the Test Sets within the Product. When a Product Instance is tested, these Scriptlets are run in the order determined by the hierarchical test structure for that Product.

This section explains in detail what can be contained in each type of Scriptlet.

[Product Scriptlet](#)

This Scriptlet will only occur ONCE during a test. It will be loaded at the start of a test, and will remain in memory for the duration of the test. It can thus contain [constants](#), [variables](#), functions and subroutines that can be used from *any* of the Test Set Scriptlets that make up the rest of the test structure.

The contents of the Product Scriptlet will be 'run' when the test starts. Global variables defined here will therefore be initialised and available for use throughout all Scriptlets, but you should not include any code unless it is contained within a function or a subroutine. If you require code to run at the start of a test, see the description of the [OnTestStart](#) function.

[Test Set Scriptlets](#)

There are likely to be many Test Set Scriptlets involved in a test. Each of these is loaded and run at the relevant point in the test, as determined by the test structure for the Product being tested. If a failure results in part of the test being repeated (See [Re-testing failures](#)), the relevant Test Set Scriptlet for that part of the test will be re-run.

Each of these Scriptlets is loaded and run to perform a particular test, or sequence of tests. It can contain variables and constants of its own, but these are *only* available for the duration of the Scriptlet (i.e. while it is running). A Test Set Scriptlet can use any functions, subroutines or variables defined in the Product Scriptlet (see above).

5.3 The User Script

AutoTestSQL allows [VBScript functions](#) and [subroutines](#) to be written that can be reused throughout all tests for all Products. Anything that is Product-specific should be put in the [Product Scriptlet](#); however there are times when functions may be useful across the entire range of Products, in which case they need to be put somewhere that can be used from any Scriptlet for any Product.

AutoTestSQL provides the **User Script** for this purpose. It is a script called **User.vbs** which resides in the Scripts folder (as defined in the [Options dialogue box](#)). Into this can be inserted any functions and subroutines that will then be available to all Test Set [Scriptlets](#).

A sample User Script is provided with AutoTestSQL, that contains some useful pre-defined functions and subroutines. For details of these, see [Functions in the User Script](#).

5.4 The SetResult function

Within a Scriptlet, [VBScript](#) can be written to set up the test equipment (for example, the dScope) and the [EUT](#). The test equipment can be queried to take measurements, and it is up to the Scriptlet to then write these measurements to the database.

The measurements are written to the database using the **SetResult** function. This function takes a result value (either a numerical value, a boolean value representing pass/fail, or a text string) and, optionally, details of which result element it is writing to (See [SetResult](#) in the [Built-in functions](#) section for a detailed description of this function's parameters).

A Scriptlet for a Test Set can use the SetResult function to write *any* of the results below it in the test structure. It can do this in one of two ways: either in the order in which they are listed (in which case the call to SetResult only needs to pass the result value itself), or in any order chosen by the script writer (in which case extra parameters need to be passed to SetResult, to indicate which result is being written).



In order for [Fault-find mode](#) to work correctly, and enable the script to be stopped in the correct state at any call to SetResult, the SetResult function must be called at the point that the measurement is made. See [Considering Fault-find mode](#) for full details.

For example, consider the following part of a test structure:

```
D/A Converter
  Audio Tests
    XLR Input Tests
      Channel 1
        Amplitude (dBu)
        THD+N (dB)
        Cross-talk (%)
      Channel 2
        Amplitude (dBu)
        THD+N (dB)
        Cross-talk (%)
    XLR Output Tests
  ..
Other Tests
..
```

We will assume for this example that the "XLR Input Tests" Test Set contains the Scriptlet whose responsibility it is to write the results to the database. There are two ways on which the results could be written:

The first way involves writing the six results to the database in the order that they are listed in the tree structure, i.e.

```
SetResult(ampl_ch1)
SetResult(thdn_ch1)
SetResult(crosstalk_ch1)
SetResult(ampl_ch2)
SetResult(thdn_ch2)
SetResult(crosstalk_ch2)
```

(Note that code would have to be written in between these calls, to actually read the results from the dScope into the variables named ampl_ch1, thdn_ch1, etc).

The second way allows us to write the results in any order that we choose. For example, using the dScope, it is possible to read both channel's results at the same time, so time would be saved by doing this within the script. In this case, we would probably read both channels' amplitude first, then the distortion, then the cross-talk. In this case the SetResult calls would be done in a different order to the order of the Test Elements in the test structure, and so we would have to supply the location of

each result as part of the call to SetResult:

```
SetResult(AMPL_ch1, 0, 0)
SetResult(AMPL_ch2, 1, 0)
SetResult(THDN_ch1, 0, 1)
SetResult(THDN_ch2, 1, 1)
SetResult(CROSSTALK_ch1, 0, 2)
SetResult(CROSSTALK_ch2, 1, 2)
```

In each of these cases, we need to pass to SetResult the zero-based indexes of the result in the structure. See [Result Indexes](#), below, for full details.

Note that the order in which the various calls to SetResult are made will have an impact on fault-find mode; see [Considering Fault-find mode](#) for further details.

[Result Indexes](#)

Each Result Element and Result Set can be thought of as having a zero-based index into its parent.

In the example above:

Within the "Channel 1" Test Set, "Amplitude (dBU)" has an index of 0, "THD+N (dB)" has an index of 1, and "Cross-talk (%)" has an index of 2.

Similarly, within the "Channel 2" Test Set, "Amplitude (dBU)" has an index of 0, "THD+N (dB)" has an index of 1, and "Cross-talk (%)" has an index of 2.

Within the "XLR Input Tests" Test Set, "Channel 1" has an index of 0, and "Channel 2" has an index of 1.

Within the "Audio Tests" Test Set, "XLR Input Tests" has an index of 0, and "XLR Output Tests" has an index of 1.

Within the "D/A Converter" Product, "Audio Tests" has an index of 0, and "Other Tests" has an index of 1.

For example: Taking the Scriptlet for the "XLR Input Tests" Test Set, to write the result for "Cross-talk (%)" in the "Channel 2" Test Set: "Channel 2" has an index of 1 within "XLR Input Tests", and "Cross-talk (%)" has an index of 2 within "Channel 2". So to write this result, we must pass parameters of 1, 2 to the SetResult function.



If the test structure changes, it is important that the relevant Scriptlets are updated to reflect any possible change in these indexes.

5.5 Considering Fault-find mode

When writing [Scriptlets](#) to be incorporated into the test structure, you will need to take into account the fact that the Scriptlets will also be run in Fault-find mode.

[Fault-find mode](#) allows you to select an individual Result Element, and run Scriptlets from the test structure up to the point where a [failed result](#) was written to the database. To do this, AutoTestSQL must stop the currently running Scriptlet at the point where [SetResult](#) is called for the offending result. This means that at the point where SetResult is called, the state of the various items of test apparatus (the [EUT](#), the dScope, [I/O switchers](#), etc) must be in the correct position to show the failure.

This means that calls to SetResult within each Scriptlet must occur at the point where the equipment is in the correct state. It would not be good practice to take several measurements involving changes to the apparatus, storing the values in [variables](#) along the way, and then call SetResult several times to actually store the results in the database. This would mean that at the time that SetResult was called, sufficient changes may have been made to put the equipment in an entirely different state. Fault-find mode would not be useful, as the point where it stopped the script may not show the problem.

As an example, consider the following part of a test structure:

```
XLR Input Tests
  Channel 1
    Amplitude (dBu)
    THD+N (dB)
  Channel 2
    Amplitude (dBu)
    THD+N (dB)
```

where the "XLR Input Tests" Test Set contains a Scriptlet that makes measurements for the four Test Elements below it in the tree.

The following Scriptlet would make the relevant measurements, and store them in the same order that they have been entered in the test structure:

```
' Make measurements...
dScope.CTDetector.CTD_Function = "Amplitude"
dAmplitudeA = dScope.CTDetector.CTD_ChA
dAmplitudeB = dScope.CTDetector.CTD_ChB
dScope.CTDetector.CTD_Function = "THD+N - relative"
dDistortionA = dScope.CTDetector.CTD_ChA
dDistortionB = dScope.CTDetector.CTD_ChB

' Then store the results in the database
SetResult(dAmplitudeA)
SetResult(dAmplitudeB)
SetResult(dDistortionA)
SetResult(dDistortionB)
```

However, if the amplitude measurement on channel B failed for some reason, Fault-find mode would run this script until it got to the second SetResult call in this Scriptlet (the one actually writing the amplitude on channel B). At this point however, the last state of the dScope was measuring THD+N, so the point at which the script stopped would not give the correct state of the apparatus at the time of failure.

The appropriate approach in this situation would be to write the results at the same time that they were read from the dScope:

```
' Make measurements, and store results at
' the same time...
dScope.CTDetector.CTD_Function = "Amplitude"
Call SetResult(dScope.CTDetector.CTD_ChA, 0, 0)
Call SetResult(dScope.CTDetector.CTD_ChB, 1, 0)
dScope.CTDetector.CTD_Function = "THD+N - relative"
Call SetResult(dScope.CTDetector.CTD_ChA, 0, 1)
Call SetResult(dScope.CTDetector.CTD_ChB, 1, 1)
```

In this case, a failure of channel B's amplitude measurement would result in Fault-find mode stopping the script at the right place, since at the time that the relevant SetResult function is called, the dScope is in the correct state.



Note that since we are no longer writing the results in the same order that they are entered in the test structure, we must use SetResult's optional zero-based index parameters to define which result we are actually writing. See [The SetResult function](#) for full details of these parameters.

5.6 Script functions

This section describes some of the built-in functionality available to Product and Test Set Scriptlets.

Certain [functions](#) and [subroutines](#) with pre-specified names can be included in your Scriptlets. If present, each one will be called at a certain time during a test - for example, when the test starts or ends. For more details, see [Optional Scriptlet functions](#).

The AutoTestSQL software exposes some functions via its [automation](#) interface. For details of these functions and how they can be used from within a Scriptlet, see [Built-in functions](#).

The [User Script](#) supplied with AutoTestSQL contains some functions provided by Prism Sound that can be used within your tests. For more details, see [Functions in the User Script](#).

5.6.1 Optional Scriptlet functions

When a test is run, the AutoTestSQL software will look for certain named [functions](#) and [subroutines](#) in your Scriptlets and, if they are present, will call them at the appropriate time during the test:

[OnTestStart](#) is called when the test is first run.

[OnTestFinish](#) is called when the test is stopped (either manually, or because the test has completed).

[ValidateSerialNumber](#) is called when the serial number of the Product Instance ([EUT](#)) is entered at the start of a test.

5.6.1.1 OnTestStart

bRet = OnTestStart ()

This function should exist in the [Product Scriptlet](#) if it is required. If it is present, it will be called when a test is first run (after validation of the serial number). It should initialise anything relevant to the test for this Product.

This function should initialise any global variables, and create any objects that will be used throughout the test.



It is imperative that any objects created within this function for the duration of the test are closed correctly in the [OnTestFinish](#) subroutine, which is run at the end of the script.

[Parameters](#)

This function has no parameters.

[Return value](#)

Your function must return **True** if it successfully performed any necessary startup, or **False** otherwise. Your function can return this value using `OnTestStart = True` or `OnTestStart = False` before exiting the function.

Note that you must return **True** to allow the test to continue running. If you return **False**, the test will stop.

Example

The following example turns dScope Settling on, and minimizes the dScope display.

```
Function OnTestStart()  
    ' Variables  
    Dim strMesg      ' Used to display messages  
  
    ' Ensure that script uses settling  
    dScope.Options.OPT_UseSettlingsFromScripts = True  
  
    ' Minimize the dScope display  
    dScope.Display(dScope.DISPLAY_MINIMIZED)  
  
    ' This routine completed successfully  
    OnTestStart = True  
  
End Function
```

5.6.1.2 OnTestFinish

OnTestFinish ()

This subroutine should exist in the [Product Scriptlet](#) if it is required. If it is present, it will be called when a test completes. This will happen whether the test stops automatically or is stopped manually by using the [Stop] button from [Test mode](#).

This subroutine should be used to clean up any objects that have been used in the script, to ensure that the test is stopped tidily. It will probably close and delete objects created in [OnTestStart](#), and any other objects that have been initialised but not closed.

Parameters

This subroutine has no parameters.

Return value

This subroutine has no return value.

Example

The following example shuts down a switcher used in tests.

```
Sub OnTestFinish()  
    Dim plStatus ' Status variable from Reset  
  
    ' Shut down switchers  
    dSNet.DSNET_Reset(0, False, plStatus)  
  
    ' Tell the Operator it's finished  
    MsgBox "Test Finished!"  
  
End Sub
```

5.6.1.3 ValidateSerialNumber

bRet = ValidateSerialNumber (strSerialNum)

This function should exist in the [User Script](#) if it is required. If it is present, it will be called when a test is first run, and the Operator enters the serial number of the Product Instance to be tested. It can display messages to the Operator if the serial number is invalid.

Parameters

strSerialNum The serial number entered by the Operator.

Return value

Your function must return **True** if the serial number is valid, or **False** otherwise. Your function can return this value by using `ValidateSerialNumber = True` or `ValidateSerialNumber = False` before exiting the function.

Note that you must return **True** to allow the test to continue running. If you return **False**, the test will stop.

Example

The following example validates the serial number, ensuring that it is 10 characters long, and that all the characters are numbers:

```
Function ValidateSerialNumber(strSerialNum)
' Vars
Dim cChar          ' current character
Dim strMsgTitle    ' Message box title

' Init vars
strMsgTitle = "Validating Serial Number..."

' Check that the serial number is 10 characters
If (Len(strSerialNum) <> 10) Then
    MsgBox "Serial number must be 10 characters!", vbOKOnly,
strMsgTitle
    Exit Function
End If

' Check that it's all numeric
For i = 1 To 10
    cChar = Mid(strSerialNum, i, 1)
    If ((cChar < "0") Or (cChar > "9")) Then
        MsgBox "Serial number must be numeric!", vbOKOnly,
strMsgTitle
        ValidateSerialNumber = False
        Exit Function
    End If
Next

' Otherwise, serial number is valid
ValidateSerialNumber = True

End Function
```

5.6.2 Built-in functions

Several [functions](#) and [subroutines](#) are exposed by the AutoTestSQL program and are available for use within your Scriptlets.

The following functions and subroutines are exposed by the AutoTestSQL application:

[SetResult](#)
[GetLimits](#)
[GetMode](#)
[GetProductFamily](#)
[GetProduct](#)
[GetSerialNumber](#)
[GetOperatorName](#)
[GetOperatorID](#)
[TestStopped](#)
[TestFailed](#)
[StopTest](#)

5.6.2.1 SetResult

bRet = SetResult (ResultValue [, iIndex1 [, iIndex2 ... iIndex10]])

This function writes a result value to the database.

For a detailed description of how this function should be used, see [The SetResult function](#).

[Parameters](#)

ResultValue	The result value to write to the database. This can be a numerical value, a boolean (True or False), or a string. Which one of these you use will depend on the result type of the corresponding Test Element (see Test Sets and Test Elements for details).
iIndex1 .. iIndex10	Optional parameters used to specify which Test Element in the test structure to set this result for. These parameters are zero-based indexes into the next level down in the test structure. See The SetResult function for more details.

[Return value](#)

This method returns **True** if the result was successfully written to the database, or **False** otherwise.

It may fail because it has been called too many times for the number of Test Elements within the Test Set containing the Scriptlet, or because the indexes do not correspond to a valid Test Element in the structure.

5.6.2.2 GetLimits

bRet = GetLimits(UpperLimit, LowerLimit [, ilIndex1 [, ilIndex2 ... ilIndex10]])

This function reads the limits for a Test Element from the database.

Scriptlets do not usually have any knowledge of [limits](#) for a test, since all this information is stored in the database and is used automatically by the AutoTestSQL application to work out the status of a result (see [How a result status is calculated](#)). However, under some circumstances, it may be useful to know this information from within a script. For example, you may want to find out the limits for a particular test, in order to show visually the limits on a dScope Reading.

This function enables the limits to be extracted from the database. Since limits correspond exactly to a specific Test Element, AutoTestSQL must be told which Test Element to extract the limits for. This function works exactly the same way as the [SetResult](#) function in this respect. It can be called without the optional index parameters, in which case it will simply get the limits for the next result in the test structure (the next result that has not yet been written using [SetResult](#)). If the optional index parameters are used, then they define exactly which Test Element in the test structure to get the limits for.

Parameters

UpperLimit

When this function returns, this variable will contain the upper limit for the corresponding Test Element.

NB: This may be a numerical value, a boolean, or a string depending on the result type of the Test Element. (see [Test Sets and Test Elements](#) for details).

LowerLimit

When this function returns, this variable will contain the lower limit for the corresponding Test Element.

NB: This may be a numerical value, a boolean, or a string depending on the result type of the Test Element. (see [Test Sets and Test Elements](#) for details)

ilIndex1 .. ilIndex10

Optional parameters used to specify which Test Element in the test structure to get the limits for. These parameters are zero-based indexes into the next level down in the test structure. See [The SetResult function](#) for more details.

Return value

This method returns **True** if the limits were retrieved successfully, or **False** otherwise.

It may fail because it has been called too many times for the number of Test Elements within the Test Set containing the Scriptlet, or because the indexes do not correspond to a valid Test Element in the structure.

Example

The following example reads a set of limits from the database and displays them on a dScope Reading.

```
' Variables
Dim dUpperLimit
Dim dLowerLimit

' Get the limits for the next result
Call GetLimits(dUpperLimit, dLowerLimit)
```

```
' Get the Reading to set details of
If dScope.GetFirstReadingForResult _
  (dScope.RESULT_CTD_CHA) Then _
  ' Turn on the bar graph...
  dScope.Reading.RDG_ShowBarGraph = True
  ' Turn limit checking on
  dScope.Reading.RDG_LimitCheckingOn
  ' Show the limits on the bar graph
  dScope.Reading.RDG_MinLimit = dLowerLimit
  dScope.Reading.RDG_MaxLimit = dUpperLimit
End If
```

5.6.2.3 GetMode

strMode = GetMode()

This function returns the current mode that AutoTestSQL is in. This can only ever be Test mode or Fault-find mode whilst a script is running.

This function may be useful if different actions need to be taken depending on whether a test is actually being performed, or whether Fault-finding is in progress. For example, you may have certain setup routines that involve asking the Operator to visually check some aspect of the EUT. This may not be relevant when running a section of the test in Fault-find mode, so you could only include these messages to the Operator during Test mode.

Parameters

This function has no parameters.

Return value

This function returns a string indicating the current AutoTestSQL mode. This will be one of the following values:

"MODE_TEST"	The script is currently running in Test mode.
"MODE_FAULTFIND"	The script is currently running in Fault-find mode.

Example

The following example conditionally displays a message if the script is running in Test mode, to make things quicker for the Operator in Fault-find mode.

```
' Variables
Dim bLEDsOK      ' True if LEDs are OK
Dim strMesg      ' Used for message

' Initialise variable - although in Fault-find
' mode it doesn't matter
bLEDsOK = True

' Do LED checks only in TEST mode
If GetMode() = "MODE_TEST" Then
  strMesg = "Are all the LEDs lit?"
  If MsgBox (strMesg, vbYesNo) = vbNo Then
    bLEDsOK = False
  End If
```



```
End If

' Set the result
' Note that we must ALWAYS do this - see note!
SetResult (bLEDsOK)
```



You must ALWAYS call SetResult whether in Test mode or Fault-find mode, even though nothing gets written to the database in Fault-find mode, since the calls to SetResult are where AutoTestSQL checks to see whether Fault-find mode should stop.

5.6.2.4 GetProductFamily

strName = GetProductFamily()

This function returns the name of the Product Family of the Product that is being tested.

Parameters

This function has no parameters.

Return value

This function returns the name of the Product Family containing the Product being tested.

5.6.2.5 GetProduct

strName = GetProduct()

This function returns the name of the Product being tested.

Parameters

This function has no parameters.

Return value

This function returns the name of the Product being tested.

5.6.2.6 GetSerialNumber

strSerialNum = GetSerialNumber()

This function returns the serial number of the Product Instance being tested, as entered by the Operator at the start of the test.

Parameters

This function has no parameters.

Return value

This function returns the serial number of the Product Instance being tested.

5.6.2.7 GetOperatorName

strName = GetOperatorName()

This function returns the name of the currently logged in Operator, i.e. the one performing the current Test or Fault-finding.

Parameters

This function has no parameters.

Return value

This function returns the name of the Operator currently logged in to AutoTestSQL.

5.6.2.8 GetOperatorID

iRet = GetOperatorID()

This function returns the ID of the currently logged in Operator, i.e. the one performing the current Test or Fault-finding.

Parameters

This function has no parameters.

Return value

This function returns the ID of the Operator currently logged in to AutoTestSQL as an integer value.

5.6.2.9 TestStopped

bRet = TestStopped()

This function returns whether the test has been stopped. This should be unnecessary in your Scriptlets, as test stopping should be managed automatically by the AutoTestSQL software.

However, there may be instances where your Scriptlets run a loop which may take some time. This function could be used to determine whether the script has been stopped, and if so, to break out of the loop.

Parameters

This function has no parameters.

Return value

This function returns **True** if the test has been stopped, or **False** if the test is still continuing.

5.6.2.10 TestFailed

bRet = TestFailed()

This function returns whether the test has failed at any point since starting.

Parameters

This function has no parameters.

Return value

This function returns **True** if any Result Elements in the current test have failed, or **False** if all Result Elements have passed so far.

5.6.2.11 StopTest

StopTest()

This function can be used to stop the test. It will stop the currently running Scriptlet, and call the [OnTestFinish](#) function in the Product Scriptlet to finish the test neatly.

Parameters

This function has no parameters.

Return value

This function has no return value.

5.6.3 Functions in the User Script

In addition to Product-specific script code in Product Scriptlets, AutoTestSQL allows functions and subroutines to be defined for use throughout the entire range of Products in a [User Script](#).

When AutoTestSQL is installed, a default version of this script is copied into the Scriptlets folder (as defined in the [Options dialogue box](#)). It contains two important functions that may be useful within your tests:

[ColourMsgBox](#) works in a similar way to the standard [VBScript MsgBox](#) function, and allows an improved, coloured background to messages to the Operator.

[FailMsgBox](#) uses a similar coloured message box that allows automatic tests of switches, pots etc to complete without requiring input from the user.

5.6.3.1 ColourMsgBox

iRet = ColourMsgBox (strMesg, vbButtons, strCaption)

This method displays a message box with a coloured background. It can be used to improve the interface shown to the Operator during tests, rather than using the standard [VBScript](#) MsgBox.

Note that the colour, position and size of this message box can be defined using other simple subroutines. See [Associated subroutines](#) below for details.

Parameters

<i>strMesg</i>	Message string to display. If strMesg consists of more than one line, you can separate the lines using a carriage return/linefeed character combination (vbCrLf) between each line.
<i>vbButtons</i>	Value specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. See the Button values section for allowed values.
<i>strCaption</i>	String displayed in the title bar of the dialogue box.

Return value

This function returns the button that was pressed by the user. See [Return values](#) for possible return values.

Button values

The **vbButtons** argument settings are:

Constant	Description
vbOKOnly	Display OK button only
vbOKCancel	Display OK and Cancel buttons.
vbAbortRetryIgnore	Display Abort , Retry , and Ignore buttons.
vbYesNoCancel	Display Yes , No , and Cancel buttons.
vbYesNo	Display Yes and No buttons.
vbRetryCancel	Display Retry and Cancel buttons.
vbCritical	Display Critical Message icon.
vbQuestion	Display Warning Query icon.
vbExclamation	Display Warning Message icon.
vbInformation	Display Information Message icon.
vbDefaultButton1	First button is default.
vbDefaultButton2	Second button is default.
vbDefaultButton3	Third button is default.
vbDefaultButton4	Fourth button is default.
vbApplicationModal	Application modal; the user must respond to the message box before continuing work in the current application.
vbSystemModal	System modal; all applications are suspended until the user responds to the message box.

These values are added together to create a value for the **vbButtons** parameter.

The first group of values (**vbOkOnly** to **vbRetryCancel**) describes the number and type of buttons displayed in the dialogue box; the second group (**vbCritical** to **vbInformation**) describes the icon style; the third group (**vbDefaultButton1** to **vbDefaultButton4**) determines which button is the default; and the fourth group (**vbApplicationModal** and **vbSystemModal**) determines the modality of the message box. When adding together to create a value for the **vbButtons** parameter, you should use only one number from each group.

Return values

Value	Description
vbOK	OK button was pressed.
vbCancel	Cancel button was pressed.
vbAbort	Abort button was pressed.
vbRetry	Retry button was pressed.
vbIgnore	Ignore button was pressed.
vbYes	Yes button was pressed.
vbNo	No button was pressed.

Associated subroutines

The following simple subroutines can be used to alter the position, size and background and text colours of the coloured message box:

SetColourMsgBoxPosition(iX, iY)

Sets the position of the coloured message box in pixels from the top left of the screen. The default position is in the centre of the screen.

SetColourMsgBoxWidth(iWidth)

Sets the width of the message box, in pixels. The default is 300 pixels.

SetColourMsgBoxTextColour(iRed, iGreen, iBlue)

Sets the colour of the text in the message box. See [Colours](#) for possible values for the iRed, iGreen and iBlue parameters. The default values are all 255, i.e. white text.

SetColourMsgBoxBkgndColour(iRed, iGreen, iBlue)

Sets the colour of the background of the message box. See [Colours](#) for possible values for the iRed, iGreen and iBlue parameters. The default values are all 0, i.e. a black background.

SetColourMsgBoxAlignment(iAlignment)

Sets the alignment of the text in the message box. See [Alignment](#) for possible values for the iAlignment parameter. The default alignment is horizontally centred at the top of the message box.

5.6.3.2 FailMsgBox

iRet = FailMsgBox (strMesg, strCaption)

This method displays a message box with a coloured background, with a [Failed] and a [Cancel] button. This allows simple tests of switches, pots etc. to be performed that can automatically detect whether a condition has been met without requiring input from the Operator. The Fail button allows the Operator to manually specify that the test has failed, if the required condition is not reached.

For example, this message box could be used to ask the user to turn on a "mute" switch. This FailMsgBox would be displayed, and the code would wait for a certain drop in amplitude. If this drop occurs, then the message box would disappear automatically without further intervention from the Operator, and the test would continue. In the event of the mute switch not working correctly, the drop in amplitude would not be detected, but the Operator could manually click the [Failed] button which would close the message box and continue with the test.

To use the FailMsgBox, you must firstly call the FailMsgBox function with the relevant message text. Your code must then loop until either the relevant condition has been met, or the FailMsgBox has been closed (use [FailMsgBox_Closed\(\)](#) to determine this). You can then use either the [FailMsgBox_Failed\(\)](#) or [FailMsgBox_Cancelled\(\)](#) function to determine whether the required condition was met, or whether the user clicked on the [Failed] or the [Cancel] button.

NB: You must always use [CloseFailMsgBox](#) to ensure that the message box is closed regardless of whether the required condition was met, or whether the Operator's button press has already closed it. See the [Example](#) below for an example of how to use the FailMsgBox.

Note that the colour, position and size of this message box can be defined using other simple subroutines. See [Associated subroutines](#) below for details.

Parameters

<i>strMesg</i>	Message string to display. If strMesg consists of more than one line, you can separate the lines using a carriage return/linefeed character combination (vbCrLf) between each line.
<i>strCaption</i>	String displayed in the title bar of the dialogue box.

Return value

This subroutine has no return value. See [Closing the FailMsgBox](#) for details of how to find out which whether the message box was closed automatically, or due to the Operator clicking on a button.

Closing the FailMsgBox

These routines allow you to close the FailMsgBox from your code, and to determine how the FailMsgBox was closed. For an example of their use, see the [Example](#) below.

CloseFailMsgBox()

Use this subroutine to close the FailMsgBox if the required condition has automatically been reached.

FailMsgBox_Closed()

Use this function to determine whether the FailMsgBox has been closed. It returns **True** if the message box has been closed (either automatically or due to the Operator clicking on a button), or **False** if it is still open.

FailMsgBox_Failed()

Use this function to determine whether the FailMsgBox has been closed due to a failure. It returns **True** if the message box was closed due to a click on the [Failed] button, or **False** if the message box is still open or has been closed in another way.

FailMsgBox_Cancelled()

Use this function to determine whether the FailMsgBox has been closed due to cancelling. It returns **True** if the message box was closed due to a click on the [Cancel] button, or **False** if the message box is still open or has been closed in another way.

Associated subroutines

The following simple subroutines can be used to alter the position, size and background and text colours of the FailMsgBox:

SetFailMsgBoxPosition(iX, iY)

Sets the position of the coloured message box in pixels from the top left of the screen. The default position is in the centre of the screen.

SetFailMsgBoxWidth(iWidth)

Sets the width of the message box, in pixels. The default is 300 pixels.

SetFailMsgBoxTextColour(iRed, iGreen, iBlue)

Sets the colour of the text in the message box. See [Colours](#) for possible values for the iRed, iGreen and iBlue parameters. The default values are all 255, i.e. white text.

SetFailMsgBoxBkgndColour(iRed, iGreen, iBlue)

Sets the colour of the background of the message box. See [Colours](#) for possible values for the iRed, iGreen and iBlue parameters. The default values are all 0, i.e. a black background.

SetFailMsgBoxAlignment(iAlignment)

Sets the alignment of the text in the message box. See [Alignment](#) for possible values for the iAlignment parameter. The default alignment is horizontally centred at the top of the message box.

Example

```

' *****
' Name      : SwitchOff
' Description: Function to determine whether a
'            switch has been muted
' Params    : eChannel      Which channel
'            : dLowerLimit   Limit to detect when muted
'            : strMsg        Message to display
'            : strCaption    Title of message
' *****
Function SwitchOff(eChannel, dLowerLimit,
                  strMsg, strCaption)
' Variables
Dim dValue      ' Amplitude value

' Get the channel
If eChannel = CHANNEL_B Then
    dValue = dScope.SignalAnalyzer.SA_ChBRMSAmpl
Else
    dValue = dScope.SignalAnalyzer.SA_ChARMSAmpl
End If

' Stop when Ampl less than limit
bAutoStop = (dValue <= dLowerLimit)

' Show the FailMsgBox
FailMsgBox strMsg, strCaption

' Wait until 'Fail' pressed, or switch muted
While ((Not bAutoStop) And _
       (Not FailMsgBox_Closed()))

    ' Re-read the amplitude
    If eChannel = CHANNEL_B Then
        dValue = dScope.SignalAnalyzer.SA_ChBRMSAmpl
    Else
        dValue = dScope.SignalAnalyzer.SA_ChARMSAmpl
    End If
    bAutoStop = (dValue <= dLowerLimit)
Wend

' Close the box (in case it wasn't
' done automatically)
CloseFailMsgBox()

' Set this result - True if signal muted,
' False if Operator selected Failed
SwitchOff = Not FailMsgBox_Failed()

End Function

```


5.6.3.3 Colours

Various methods in the [User Script](#) allow you to set colours by passing a red, green and blue component. These combine to make a single colour.

Here are some examples of how these components combine to make common colours:

Colour	Red	Green	Blue
White	255	255	255
Black	0	0	0
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Magenta	255	0	255
Yellow	255	255	0
Cyan	0	255	255
Light Grey	192	192	192
Dark Grey	128	128	128
Orange	255	128	0

For further colours, you can use the windows colour palette. This palette is used in several programs, including the dScope, and can be accessed by right-clicking on the Trace legend, selecting "Change Trace colour" and clicking on "Define Custom Colours". This will give you a palette of colours; clicking on any one of these will show you the red, green and blue components of that colour.

5.6.3.4 Alignment

Some functions in the [User Script](#) allow you to set the alignment of text in a control. For full details of this, see the manual for the ScriptDlg ActiveX Control that is provided with the dScope software.

Alignment of text can be a combination of a horizontal alignment and a vertical alignment. (Note that if more than one horizontal alignment is specified, the first one from the list below will be used. Similarly, if more than one vertical alignment is specified, the first one will be used).

[Values](#)

alignLeft	1	Sets the message box text to be horizontally aligned to the left of the control.
alignHCentre	2	Sets the message box text to be horizontally aligned in the centre of the control.
alignRight	4	Sets the message box text to be horizontally aligned to the right of the control.
alignTop	8	Sets the message box text to be vertically aligned at the top of the control.
alignVCentre	16	Sets the message box text to be vertically aligned in the centre of the control.
alignBottom	32	Sets the message box text to be vertically aligned at the bottom of the control.

5.7 Writing and debugging scripts

This section provides information on writing and debugging your scripts in AutoTestSQL.

To find out information on choosing a text editor for your scripts, see [Choosing a script editor](#).

For some tips on writing scripts, see [Writing scripts](#).

For some tips on debugging the scripts that you have written, see [Debugging scripts](#).

For a list of some common problems encountered when writing scripts for AutoTestSQL, see [Common script problems](#).

For an overview of how to make your scripts as reusable as possible, see [Reusability of scripts](#).

5.7.1 Choosing a script editor

In [Setup mode](#), clicking the [View/Edit] button when a [Scriptlet](#) is associated with a Product or Test Set will result in the Scriptlet being opened in [Notepad](#). This is a very simple text editor and does not provide any colouring of [VBScript](#) or dScope keywords, and so relies very much on the script writer not making any mistakes.

A much better option is to use an editor which can be made aware of VBScript syntax. There are several freeware editors available that can be found from a quick search on the internet. An example is the Crimson Editor, which can also be set up to recognise dScope keywords as well as standard VBScript keywords - visit www.crimsoneditor.com for more details.

[Using the dScope script editor](#)

The dScope software has a built-in script editor that can be used to write Automation scripts for the dScope. It can also be used to write simple Scriptlets to attach to Products and Test Sets. It can be accessed from the "Edit script" option on the dScope software's Automation menu.

The dScope script editor displays a tree of the available properties and methods, allowing scripts to be created quickly and easily by dragging items across from the tree into the script. The relevant code is inserted without the writer needing exact knowledge of the required syntax.

This script editor has the advantage that any scripts written in this way can be run from within the dScope, thus allowing writing *and* testing of scripts before using the scripts within AutoTestSQL.

The disadvantage of this editor is that since it is built in to dScope, it does not prefix the dScope's properties and methods with "dScope.". However this can be worked around by inserting the following code at the top of the script in the Script editor:

```
Dim dScope
Set dScope = CreateObject("dScope.Application")
```

Using this code will mean that any items dragged into the script from the tree of properties and methods will be prefixed correctly with "dScope.".

Note that most Scriptlets will also use the [SetResult](#) function. Since this function is built in to AutoTestSQL, it will not be recognised by the dScope and so you will need to insert a dummy version at the top of the script if you wish to use the dScope to run and debug these Scriptlets as well. The same applies to any other functions that you will need to use from within your Scriptlets. A simple example of this function is shown below; note that this shows the value passed to it and can be used for simple debugging purposes:

```
' Dummy SetResult function
Function SetResult(Val)
    MsgBox "SetResult called with a value of " & Val
    SetResult = True
End Function
```



You will need to ensure that any code added to the script in this way is removed before the Scriptlet is used with AutoTestSQL.

5.7.2 Writing scripts

When writing scripts, there are some steps that you can take to make life easier for yourself and ensure that any script problems are easier to debug. A discussion of programming principles is beyond the scope of this manual; however some ideas are outlined below:

Readability of Scriptlets

It is important to make your Scriptlets as readable as possible, to enable you to see easily how the code works. This will be invaluable if you need to come back to code at a later date and change it.

Code style

It is often a good idea to have a particular style of coding, and stick to that style throughout your Scriptlets - for example, how you indent lines, how you comment functions and subroutines, etc. should stay consistent throughout the code.

Comments

This is probably the single most important part of writing good Scriptlets. It is imperative that any code you write is well commented, to enable you to see at a glance what a piece of code is supposed to do without having to work it out from the VBScript itself. It also enables anyone else who looks at the code to see easily what it does - a must for projects involving more than one programmer.

Constants

If you have numbers or strings in your script that are used throughout the script, you should consider making them into [constants](#). This means that they can be defined once at the start of the script, and then reused throughout. If you decide to change the value of the constant, it only needs to be changed in one place.

For example, you may decide that you want all your messages to the Operator to have a certain caption: "My Tests". Rather than type this text as the third parameter to `MsgBox` or [ColourMsgBox](#), you would simply define a constant at the top of the script:

```
Const strMessageTitle = "My Tests"
```

and then use this through the code:

```
ColourMsgBox("Message...", vbOKOnly, strMessageTitle)
```

If at any point in the future you decide to change this caption, it is a simple matter of changing it in one place, rather than trawling through the script to find all the places that you display messages to the Operator.

Variable naming

When using variables throughout your script, ensure that they have sensible and meaningful names. You can also use a naming system such as [Hungarian notation](#), where the type of the variable ([boolean](#), [integer](#), etc) is part of the variable name.

Option Explicit

It is a good idea to always use the VBScript statement **Option Explicit** at the top of a script. This forces you to declare any variables that you will use in the script, and means that you are less likely to use an incorrect variable name by spelling it wrongly later in the script. If you do not do this, then a variable that has not been defined will always be treated as a zero. This can lead to problems that are very difficult to find later on.

AutoTestSQL includes this statement by default in the AutoTest.vbs script that is loaded at the beginning of every test. We recommend that it is not removed.

Reusability

Try to keep your code as reusable as possible, by putting common operations into [functions](#) or [subroutines](#). For more details on how to make your AutoTestSQL Scriptlets reusable, see [Reusability of scripts](#).

Checking variables

It is a good idea to check values when they are passed to a function or subroutine, to see if they are valid. This enables a Scriptlet to throw up a message when invalid values are used, and ensures that the Operator knows what the problem is. It should mean that most potential problems are trapped early on in the development process.

```
Function CreateFolder(strFolderName)
    ' Check name passed
    If strFolderName = "" Then
        MsgBox "Invalid folder name passed to CreateFolder!"
        CreateFolder = False
        Exit Function
    End If

    ' Create the folder
    ...
    CreateFolder = True
End Function
```

5.7.3 Debugging scripts

It is very unusual to write a Scriptlet and have it work correctly first time! If it does not, then you will need to do some debugging to work out where the problem lies. A full discussion of script debugging is beyond the scope of this manual; however this section gives some helpful ideas on how to start looking for problems.

Types of error

In general there are three possible kinds of error that can occur when writing scripts. See [Common script problems](#) for some examples of these.

Syntax errors

These kinds of errors are simple mistakes in the Scriptlet where the syntax of the VBScript is incorrect. These will usually be picked up when you first run the Scriptlet, since the error in the code will mean that it cannot run. An error message will usually be shown, giving the line number in the script where the problem has occurred.

Run-time errors

These are errors where the syntax of the code is correct, and the Scriptlet runs, but encounters an error during running due to an incorrect operation being performed. For example, "Division by zero" is a common run-time error - where the script tries to divide by a [variable](#) which contains the number 0. Again, an error message will usually be shown here with the offending line number.

Incorrect operation

These are often the most difficult errors to find, since they do not result in error messages. The script simply does not work correctly. See some of the [simple debugging techniques](#) below for some ways of finding these problems.

Simple debugging techniques

This section describes some simple techniques that can be used to locate bugs:

Using line numbers

Error messages usually give a line number where the error has occurred. It is a simple matter to use a text editor to move directly to this line number and look for the problem.

In some cases, the line number may not actually be the position of an invalid piece of code, but rather where the script engine expects the valid piece of code to be. For example, if you have missed an "End If" from the end of an "If ... Then" statement, the line number will probably not bring up the line where you would expect this statement to be, but would be the next piece of code in the Scriptlet. This is because it is only when it gets to the next bit of code that the script realizes that it's not the "End If" that it is expecting.

Take note of error messages

Most syntax errors and run-time errors will give you an error message when they occur. Occasionally this is an obscure message, but usually it is sensible and helps to explain the problem. The [Documentation on Windows Script Technologies](#), available from Microsoft, contains some useful information about VBScript run-time and syntax errors, and some ways of solving them.

Message boxes

A simple way of working out where a Scriptlet is going wrong is to insert simple "MsgBox" commands at intervals throughout a Scriptlet, displaying appropriate information to the Operator. For example, if the value of a variable is changing to an incorrect value, you could insert message boxes displaying the value of the variable at various points in the code. It should be a simple matter to track down the point at which it is going wrong.

NB: Don't forget to remove these message boxes once you find the problem!

Breaking down the problem

Finding bugs is often easier if you break the problem down. If you have a problem in a Scriptlet that performs several tasks, go through each of the tasks individually and solve the problems in each task separately. Even within an individual task, you can break the problem down until you reach the solution.

For example, if you have a result that keeps failing, you could ask yourself:

- What value is actually being passed to the SetResult function? (Insert a MsgBox to find out). Is this value correct?
- If not, then where is the result coming from?
- When the value is read from the dScope, is it correct at this point?
- If not, then is the dScope set up correctly?

Talk yourself through the problem

If you have a problem that is proving difficult to solve, it can sometimes be useful to just look at the code, and step through it line by line. At each step, make sure you know what the value of each variable should be, and how each constituent part of the test (the dScope, the [EUT](#), any [I/O switchers](#)) are set up. Sometimes just talking through the code as if you are telling someone else how it works can help you see the problem instantly!

5.7.4 Common script problems

The [Documentation on Windows Script Technologies](#), available from Microsoft, contains some useful information about VBScript run-time and syntax errors, and some ways of solving them. This should be used if you require a comprehensive reference. Some problems that do not produce error messages are listed below:

Hanging scripts

If the script stops responding, it may well be stuck in a loop and unable to finish. Check any While ... Wend loops and ensure that the relevant variable is being changed within the loop to enable it to finish at some point.

For example, the following code will cause the script to stop responding, since the value of the loop variable i is never incremented:

```
i = 1
While i <= 8
    ' Change the switcher channel
    IOSwitcher.SWITCHER_ExclusiveChannel "Inputs", i
    ' Make the measurements
    MeasureInputs()
Wend
```

Qualifying dScope constants

A common cause of problems in AutoTestSQL is forgetting to qualify dScope constants using the "dScope." prefix.

If you forget to qualify a dScope property or method, this will usually be picked up when the script is run. You will get a "Variable is undefined" error if the `Option Explicit` statement is used, or an "Object required" error otherwise.

For constants however, VBScript has the annoying habit of treating as zero those for which it cannot find a definition. This may result in an error that is exceedingly difficult to find, for example setting the dScope's Analogue Outputs to "Unbalanced":

```
dScope.AnalogueOutputs.AO_Output = AO_OUTPUT_UNBAL
```

This code will treat the `AO_OUTPUT_UNBAL` variable as zero, since it is not prefixed with the "dScope." prefix. However, zero *is* actually a valid value for the Analogue Outputs' AO_Output property, and in fact represents a *Balanced* output - the opposite of the intended effect!



Using the dScope script editor to write AutoTestSQL Scriptlets will cause this prefix to be automatically added when constants are inserted into the script. See [Using the dScope script editor](#) for details.

5.7.5 Reusability of scripts

It is important when writing Scriptlets for AutoTestSQL to ensure that they are as reusable as possible. This is achievable using [functions and subroutines](#), and also by [re-using scripts between Products](#).

Functions and subroutines

Any group of operations that may be used more than once should be put into a [function](#) or a [subroutine](#). This can then be called from somewhere else in the code to perform the same operations. This means that if a change is required to this group of operations, a single change is required to affect everywhere that it is used, rather than having to change each instance in the code separately.

Note that functions and subroutines that are available to *all* Test Set Scriptlets must be defined in the [Product Scriptlet](#) (for reuse within the tests for this Product) or in the [User Script](#) (for reuse in any test for any Product). If a function or subroutine is defined in a Test Set Scriptlet, it will *only* be available throughout that Scriptlet, and not in any others.

Reusability of scripts between Products

Under some of circumstances, the same Scriptlets may be able to be reused between Products. This is particularly likely between Products in the same Product Family. For example, you may have a range of Products where the basic functionality is the same, but the number of channels is different for each Product. You could use the same Scriptlets for some of the tests; within each Scriptlet you could use the [GetProduct](#) function to determine which Product is currently being tested, and then loop through the channels the required number of times for this Product.

Part



6

Operation reference

6 Operation reference

The operation reference section provides detailed descriptions of all parts of the AutoTestSQL software.

The main sections are as follows:

Menus	Contains details of all the menu options in AutoTestSQL.
Toolbar	Contains a description of the functions available on the AutoTestSQL Toolbar.
Setup mode	Describes Setup mode, and how it is used to set up the test database.
Test mode	Describes how Test mode is used to perform tests.
Fault-find mode	Gives details of how Fault-find mode can be used to pin-point and resolve failures in tests.
Options	Describes the Options dialogue box, and the customizable areas of AutoTestSQL.
AutoTestSQL security	Describes how security options can be used to restrict access to parts of the AutoTestSQL system for certain Operators.

6.1 Menus

The following menu options are available in the AutoTestSQL software:

[File menu](#)

Print	Prints the current page. This is context-sensitive and depends on the current mode of operation: Setup mode: Not available. Test & Fault-find mode: Prints out the test report for the current Test Session.
Print Preview	Allows viewing of the current details as they would appear on a printer. It is context-sensitive as for the Print option, above.
Print Setup...	Allows specification of printer details.
Exit	Exits the AutoTestSQL application.

[Edit menu](#)

Undo	This option is currently unavailable in AutoTestSQL.
Cut	Deletes the currently selected text or object and copies it to the clipboard.
Copy	Copies the currently selected text or object to the clipboard.
Paste	Inserts the text or object in the clipboard at the current cursor position.

View menu

The View menu allows the basic elements of the AutoTestSQL user interface to be displayed or hidden as required.

Toolbar	Toggles the AutoTestSQL Toolbar on and off.
Status bar	Toggles the Status bar on and off.

Tools menu

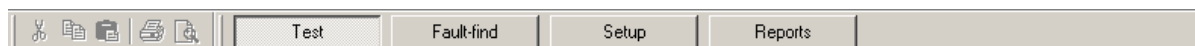
Options...	Opens the Options dialogue box , from which you can change various operating options for AutoTestSQL.
Change Operator...	Allows the currently logged-on Operator to be changed.

Help menu

Help Contents	Accesses the contents page of the on-line help file.
Help Index	Accesses the index page of the on-line help file.
Help Search	Accesses the search page of the on-line help file.
About AutoTestSQL...	Displays a box describing the current AutoTestSQL software release.

6.2 Toolbar

The main Toolbar of the AutoTestSQL application consists of two parts:



Standard buttons

Cut	Deletes the currently selected text or object and copies it to the clipboard.
Copy	Copies the currently selected text or object to the clipboard.
Paste	Inserts the text or object in the clipboard at the current cursor position.
Print	Prints the current page. This is context-sensitive and depends on the current mode of operation: Setup mode: Not available. Test & Fault-find mode: Prints out the test report for the current Test Session.
Print Preview	Allows viewing of the current details as they would appear on a printer. It is context-sensitive as for the Print option, above.

AutoTestSQL mode buttons

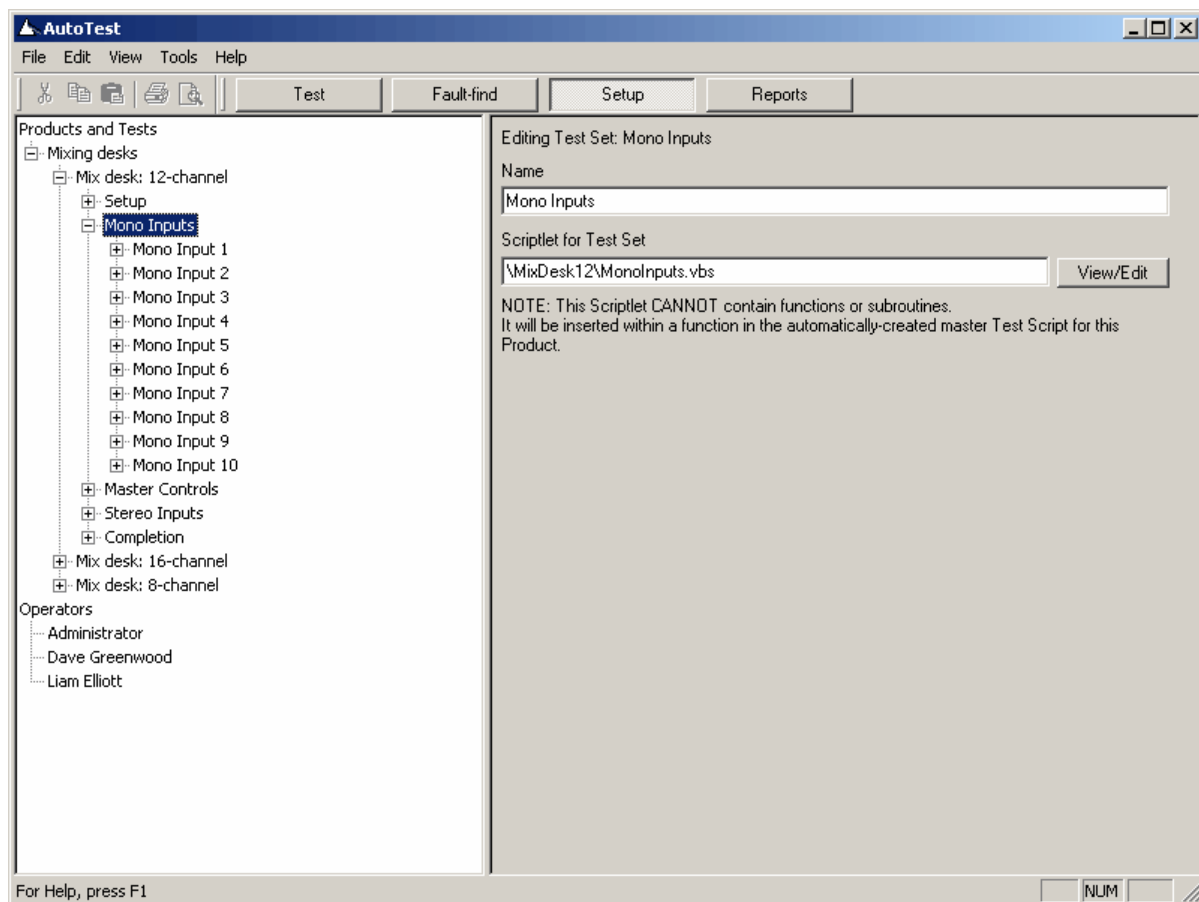
These buttons switch between the various modes of AutoTestSQL:

Test	Enters Test mode , to allow tests to be run and results to be gathered.
Fault-find	Enters Fault-find mode , which allows the cause of a failure to be pin-pointed and details logged to the database.
Setup	Enters Setup mode , allowing entry or editing of new Products, Operators and Tests.
Reports	Runs the application used to produce management reports , as specified in the Options dialogue box .

6.3 Setup mode

AutoTestSQL's **Setup mode** is used to initially define the [Products](#) that will be tested, and the tests that will be performed on them. It allows a hierarchical structure of tests to be set up for each Product, and these tests to be associated with [Scriptlets](#) that will be used to run the tests. It also allows setup of the different Operators that will use the system, and allows specification of the different tasks that each one can perform.

The screen in Setup mode consists of a hierarchical tree of items on the left hand side, together with a context-sensitive pane on the right containing the details of the currently selected tree item:



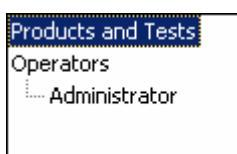
The following sections describe various operations possible in Setup mode:

[Setting up Products](#)
[Setting up tests](#)
[Setting up Operators](#)

6.3.1 Setting up Products

Before you can set up Tests to be done, you must have set up the Products that are to be tested. This is done from Setup mode. Note that you must have an adequate security level to use Setup mode; see [AutoTestSQL security](#) for details.

Once in Setup mode, the left-hand pane of the AutoTestSQL screen shows the current Products, Tests and Operators. When you first start using AutoTestSQL, there will only be two entries in this list: **Products and Tests**, and **Operators**. You will set up new Products under **Products and Tests**.



Note that if the current Operator's security level is not high enough, then the list of Products and Tests will not be available.

[Product Families](#)

Products are arranged in categories called [Product Families](#), and you will need to create at least one Product Family before you can enter any Products. To do this, right-click on the **Products and Tests** item in the tree. You will then see the following options:

New Product Family	Select this option to create a new Product Family. This Product Family will be created in the tree, and the right-hand side of the screen will change to show the details of that Product Family. For full details of entering and editing a Product Family, see Entering/editing Product Family details .
---------------------------	--

[Products](#)

Once you have a Product Family in the tree, you can right-click on it to bring up the following menu:

Delete Product Family	Select this option to delete the currently selected Product Family. Note that if the Product Family contains any Products, and any of the Products have Test Sets, then deletion will be disallowed; You must remove the Test Sets first. See Setting up tests to find out about deleting Test Sets.
New Product	Select this option to create a new Product. This Product will be created within the selected Product Family in the tree, and the right-hand side of the screen will change to show the details of that Product. For full details of entering and editing a Product, see Entering/editing Product details .

If you right-click on a Product in the tree, you will be presented with the following menu:

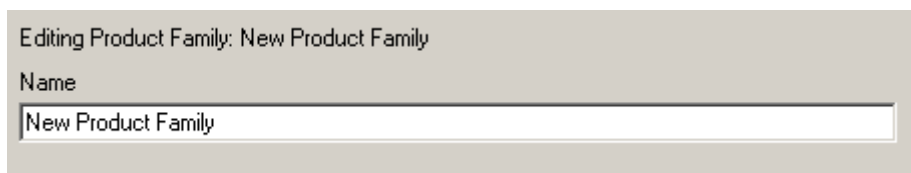
- | | |
|-----------------------|---|
| Delete Product | Select this option to delete the currently selected Product. Note that if the Product contains any Test Sets, then deletion will be disallowed; You must remove the Test Sets first. See Setting up tests to find out about deleting Test Sets. |
| New Test Set | This option allows you to set up Tests for the selected Product. See Setting up tests to find out about setting up Tests for a Product. |

Manipulating Products and Product Families

It is not currently possible to move or copy Products from one Family to another.

6.3.1.1 Entering/editing Product Family details

In Setup mode, when a new Product Family is created, or an existing Product Family is selected, the following details appear in the right-hand pane of the AutoTestSQL main window:



Editing Product Family: New Product Family

Name

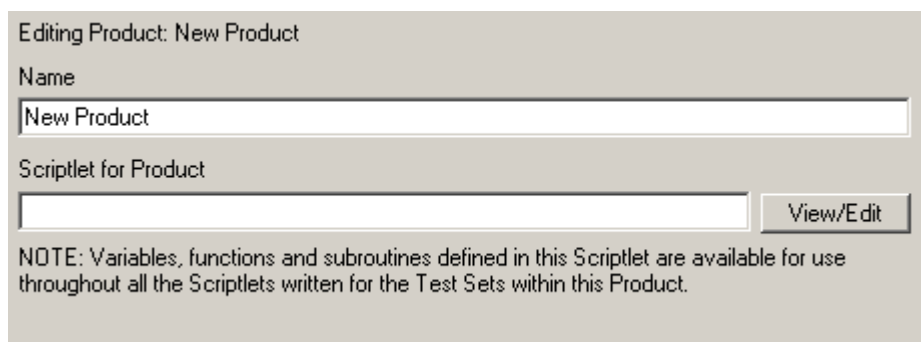
New Product Family

Fields and controls

- | | |
|-------------|---|
| Name | Specifies the name of the Product Family. This is set to "New Product Family" by default.
This name cannot contain any backslash (' \ ') characters. |
|-------------|---|

6.3.1.2 Entering/editing Product details

In Setup mode, when a new Product is created, or an existing Product is selected, the following details appear in the right-hand pane of the AutoTestSQL main window:



Editing Product: New Product

Name

New Product

Scriptlet for Product

View/Edit

NOTE: Variables, functions and subroutines defined in this Scriptlet are available for use throughout all the Scriptlets written for the Test Sets within this Product.

Fields and controls

Name	Specifies the name of the Product. This is set to "New Product" by default. This name cannot contain any backslash (' \ ') characters.
Scriptlet for Product	This specifies an optional Scriptlet containing code that can be associated with this Product. See Product and Test Set Scriptlets for full details.
View/Edit	Allows you to view or edit the Scriptlet associated with this Product.



The Note below the 'Scriptlet for Product' field is a reminder about the differences between a Scriptlet associated with a Product, and the Scriptlets associated with the individual Test Sets for the Product. See the section on [Product and Test Set Scriptlets](#) for a full explanation of these differences.

6.3.2 Setting up tests

Before you can set up tests to be done for a Product, you must have set up some Products to be tested. See [Setting up Products](#) to find out more about setting up Products for testing.

Once you have at least one Product in the setup, you can begin to enter test details. Note that you must have an adequate security level to use Setup mode; see [AutoTestSQL security](#) for details.



It is important that you have considered the structure of your tests in some details before entering them into AutoTestSQL. Please see the sections on [Test structure](#) and [Test Scriptlets](#) before entering details of any tests, as these will give you a better idea about designing your test and script structure.

Tests are set up under Products in Setup mode. Note that individual Test Elements are grouped within Test Sets; these Test Sets can be hierarchically arranged within other Test Sets.

Test Sets

To enter a new Test Set, simply right-click on the Product or the Test Set that you wish to enter the Test Set within. You will see a menu with the available options (Note that if you have clicked on a Product, only the **New Test Set** option will be available; if you have clicked on a Test Set, all the following options will be shown).

Move Test Set up	Select this option to move the currently selected Test Set up one place in the tree, within its parent Test Set or Product. This option will be disabled if the currently selected Test Set is the first one under its parent in the tree.
Move Test Set down	Select this option to move the currently selected Test Set down one place in the tree, within its parent Test Set or Product. This option will be disabled if the currently selected Test Set is the last one under its parent in the tree.
Copy Test Set	Copies the currently selected Test Set and places the copy at the bottom of the list of Test Sets within the current parent. The Test Set will be named "Copy of <name>", where <name> is the name of the currently selected Test Set.
Delete Test Set	<p>Deletes the currently selected Test Set. You will be asked to confirm deletion before it occurs.</p> <p>NB: If this Test Set has been used in any Tests, it will not actually be deleted from the database, since its details are still required by any attached Result Sets. However, it will be lost from the tree of Test Sets and cannot be recovered.</p> <p>Please see Notes about moving and deleting tests, below, for additional information.</p>
New Test Element	Select this option to create a new Test Element within the selected Product or Test Set. A new Test Element will be created in the tree, and the right-hand side of the screen will change to show the details of that Test Element. For full details of entering or editing a Test Element, see Entering/editing Test Element details .
New Test Set	Select this option to create a new Test Set within the selected Product or Test Set. A new Test Set will be created in the tree, and the right-hand side of the screen will change to show the details of that Test Set. For full details of entering or editing a Test Set, see Entering/editing Test Set details .

Test Elements

Once you have a Test Element in the tree, you can right-click on it to bring up the following menu:

Move Test Element up	Select this option to move the currently selected Test Element up one place in the tree, within its parent Test Set. This option will be disabled if the currently selected Test Element is the first one in its Test Set.
Move Test Element down	Select this option to move the currently selected Test Element down one place in the tree, within its parent Test Element or Product. This option will be disabled if the currently selected Test Element is the last one in its Test Set.
Copy Test Element	Copies the currently selected Test Element and places the copy at the bottom of the list of Test Elements within its parent Test Set. The Test Element will be named "Copy of <description>", where <description> is the description of the currently selected Test Element.
Delete Test Element	<p>Deletes the currently selected Test Element. You will be asked to confirm deletion before it occurs.</p> <p>NB: If this Test Element has been used in any Tests, it will not actually be deleted from the database, since its details are still required by any attached Result Elements. However, it will be lost from the tree of Test Sets and Test Elements and cannot be recovered.</p> <p>Please see Notes about moving and deleting tests, below, for additional information.</p>

Manipulating Test Sets and Test Elements

Test Sets and Test Elements can be moved or copied within the tree. This is particularly useful, for example, for audio devices with several similar channels. The first channel can be set up in the tree, and then subsequent channels can simply be copied from the first channel, making the process of setting them up much easier.

Test Sets and Test Elements can be copied within a parent Product or Test Set as described above (i.e. by right-clicking on the item and selecting the "Copy..." menu option). It is also possible to move or copy a Test Set or Test Element by dragging it with the mouse into a new parent. Holding down the <Ctrl> key whilst dragging will cause the item to be copied, otherwise it will be moved. You will be asked to confirm the moving or copying before it happens.

During dragging this way, the cursor will change to one of the following:



Notes about moving and deleting tests

If you wish to restructure some tests, it is worth bearing in mind how this will affect existing tests. Existing sets of results rely on the sets of test details that they were created from to get certain information - for example, the name of a Test Set, or the limits applied at test time. Under some circumstances, changing the test structure can adversely affect the results of existing tests.

As an example of this, consider a report that you may wish to produce that gives the range of results for a particular test - for example, the range of results for a "Distortion" test. The report would ask the database for all the results linked to a certain Test Element. If you were to move this Test Element in the test hierarchy, or delete it and then re-create a similar test somewhere else, then the range of results returned by the report would be different for all the tests done *before* the change, and all the tests done *after* the change.

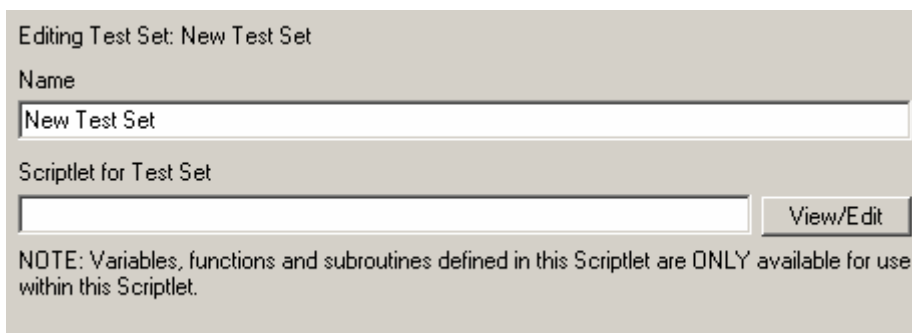
For a detailed discussion of the AutoTestSQL database structure, see [The AutoTestSQL database](#).



Once you have started entering test details in the database, it is important to back up the database regularly. To find out how to back up the database, see [Backing up and restoring the database](#).

6.3.2.1 Entering/editing Test Set details

In Setup mode, when a new Test Set is created, or an existing Test Set is selected, the following details appear in the right-hand pane of the AutoTestSQL main window:



Editing Test Set: New Test Set

Name

New Test Set

Scriptlet for Test Set

View/Edit

NOTE: Variables, functions and subroutines defined in this Scriptlet are ONLY available for use within this Scriptlet.

Fields and controls

Name	Specifies the name of the Test Set. This is set to "New Test Set" by default. This name cannot contain any backslash (' \ ') characters.
Scriptlet for Test Set	This specifies an optional Scriptlet containing code that can be associated with this Test Set. See Product and Test Set Scriptlets for full details.
View/Edit	Allows you to view or edit the Scriptlet associated with this Test Set.



The Note below the 'Scriptlet for Test Set' field is a reminder about the differences between a Scriptlet associated with a Test Set, and the Scriptlets associated with the Products themselves. See the section on [Product and Test Set Scriptlets](#) for a full explanation of these differences.

6.3.2.2 Entering/editing Test Element details

In Setup mode, when a new Test Element is created, or an existing Test Element is selected, the following details appear in the right-hand pane of the AutoTestSQL main window:

Fields and controls

Description	Specifies the description of the Test Element. This is set to "New Test Element" by default. If this Test Element is to have a numerical value, then you should define the unit that this numerical value represents as part of the test description (for example, "Amplitude (dBu)". This description cannot contain any backslash (' \ ') characters.
Result type	This specifies the type of result. See Result types below for a list of possible result types.
Use ... decimal places	This specifies how many decimal places to use when displaying this Test Element's result in the list of results. This can be a number between 0 and 15.
Upper Limit	This specifies the upper limit for this Test Element. If the result for this Test Element is above this limit, it will be stored as Failed in the database. See How a result status is calculated for further details.
Lower Limit	This specifies the lower limit for this Test Element. If the result for this Test Element is above this limit, it will be stored as Failed in the database. See How a result status is calculated for further details.

Result types

The following result types can be selected for a Test Element:

Numerical value	The result is a double-precision number. The corresponding Result Element's value must be between the Upper and Lower Limits to be classed as Passed ; otherwise it is classed as Failed .
Pass/Fail	The result can only have one of two values: a pass or a failure. From a Scriptlet, the result can be set by passing True or False, or a string representing a pass or failure (for example, "Pass"/"Fail", or "Yes"/"No"). See the reference for SetResult for full details. The Upper and Lower Limit are irrelevant for this type of result, as the status is implicit in the result value.
String	The result is a string. If neither an Upper or Lower limit is specified, this result will be classed as Non-critical and will show up as neither a pass or failure in the test results (i.e. its status will not affect the overall status of the test). If an Upper or Lower limit can be specified, the result will be classed as Passed if it matches one of the limit strings specified, or Failed if at least one limit is specified, and the result does not match it.

For full details of how a result's status is calculated for each of these result types, see [How a result status is calculated](#).

6.3.3 Setting up Operators

Setup mode allows you to enter details of all the Operators that will use AutoTestSQL. This allows you to keep track of who has performed tests and who has entered fault details in Fault-find mode.

In setup mode, the list of Operators is available under the list of Products and Tests:



Note that if the current Operator's security level is not high enough, then the list of Operators will not be available.

Note that there is always an Administrator Operator, built into AutoTestSQL, which has full access to any parts of the system. Further details of this Operator can be found under [The Administrator Operator](#) in the [AutoTestSQL security](#) section of this manual.

Setting up Operators is similar to setting up new Products. To add an Operator, simply right-click on the **Operators** item in the tree, and you will see a menu with the following options:

New Operator	Select this option to create a new Operator. This Operator will be created in the tree, and the right-hand side of the screen will change to show the details of the new Operator. For full details of entering and editing an Operator, see Entering/editing Operator details .
---------------------	--

Once you have one or more Operators in the list, you can right-click on an Operator to bring up a menu with the following options:

Delete Operator

Deletes the currently selected Operator. You will be asked to confirm deletion before it occurs.

NB: If this Operator has performed any Test Sessions, then deletion will be disallowed.



The 'Delete Operator' menu option is not available for the built-in Administrator Operator.

6.3.3.1 Entering/editing Operator Details

In Setup mode, when a new Operator is created, or an existing Operator is selected, the following details appear in the right-hand pane of the AutoTestSQL main window:

A screenshot of a software window titled "Editing Operator: New Operator". It contains several input fields and checkboxes. The "Name" field is set to "New Operator". The "ID" field is set to "1000". There is a checked checkbox labeled "This Operator must change password at next login.". Below this, a section titled "This Operator can:" contains a list of permissions with checkboxes: "Perform Tests" (checked), "Perform Fault-finding" (checked), "Access Setup mode" (unchecked), "View management reports" (unchecked), "Edit Options settings" (unchecked), "Set up Products and Tests" (unchecked), and "Set up Operators" (unchecked).

Fields and controls

- Name** Specifies the name of the Operator. This is set to "New Operator" by default.
- ID** This description cannot contain any backslash (' \ ') characters. This specifies a unique ID number for this Operator. When a list of Operators is available, for example in the [Login dialogue box](#), this number can be typed as a quick way of entering the current Operator, rather than having to select the Operator from a list.
- This Operator must change password at next login** If this box is checked, then the Operator must change their password the next time they log in to AutoTestSQL. This allows an [Administrator](#) to force Operators to change their password for security reasons. This option is checked by default, to force all Operators to enter a password when they first start using the system. This is because passwords are blank by default.
- This Operator can:** Specifies a list of possible operations whose access can be restricted to different Operators. See the list of [allowed operations](#) for full details. To allow an option, check the option in the list. To stop this option being available to this Operator, uncheck the box.
- NB:** This list is disabled for the [built-in Administrator Operator](#), and all the options are checked. This is to ensure that there is always at least one Operator on the system with full access.

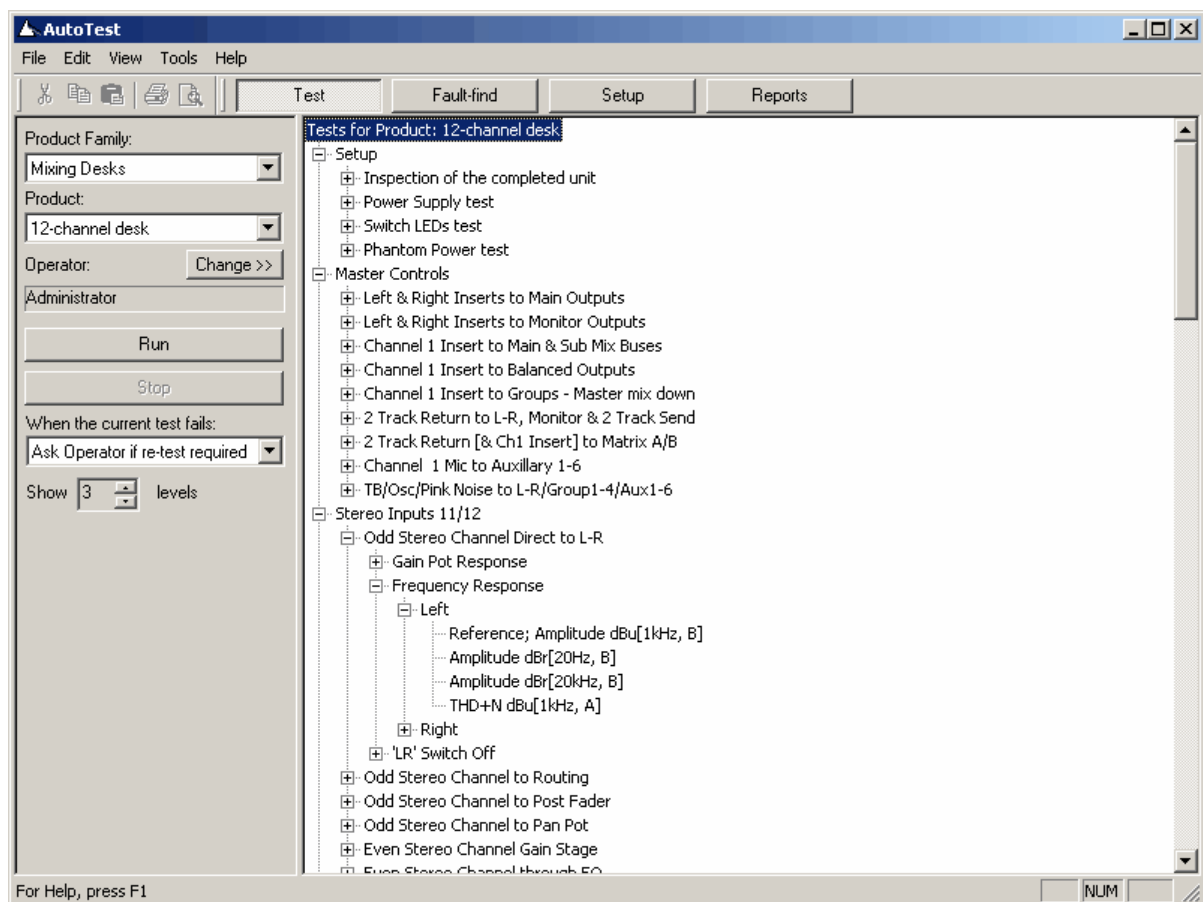
Allowed operations

Edit Options settings	Allows access to the Options dialogue box on the Utility menu, which allows changing of certain operating parameters for AutoTestSQL.
Perform Tests	Allows the Operator to perform tests using Test mode .
Perform Fault-finding	Allows the Operator access to Fault-find mode .
Access Setup mode	Allows the Operator access to Setup mode .
View management reports	This allows the Operator to access management reports from the Reports button.
Set up Products and Tests	This allows the Operator to set up Products and Tests in Setup mode .
Set up Operators	This allows the user to set up Operators in Setup mode .

6.4 Test mode

Test mode is the main part of AutoTestSQL. It is the mode that will be most commonly used on a day-to-day basis, and allows you to run tests on a Product at the click of a button.

The screen in Test mode consists of a series of fields on the left hand side, and a hierarchical tree of results on the right. This tree of results corresponds to the tests that are performed on the selected Product, and is filled in dynamically as tests are performed.



The following sections describe the operation of Test mode:

[Fields and controls](#)

[Tree of results](#)

[Running a test](#)

[Stopping a test](#)

[Test completion](#)

[The Test Report](#)

NB:

Once you have started to run tests and have some test results in the database, it is important to back up the database regularly. To find out how to back up the database, see [Backing up and restoring the database](#).

6.4.1 Fields and controls

In Test mode, the left-hand side of the screen contains the various fields and controls available:

The screenshot shows a control panel with the following elements:

- Product Family:** A dropdown menu currently showing "Mixing Desks".
- Product:** A dropdown menu currently showing "12-channel desk".
- Operator:** A text field showing "Administrator" with a "Change >>" button to its right.
- Run/Stop buttons:** Two large buttons labeled "Run" and "Stop".
- When the current test fails:** A dropdown menu currently showing "Ask Operator if re-test required".
- Show levels:** A label "Show" followed by a numeric spinner set to "3" and the word "levels".

The following is a list of the available fields and controls, and their function:

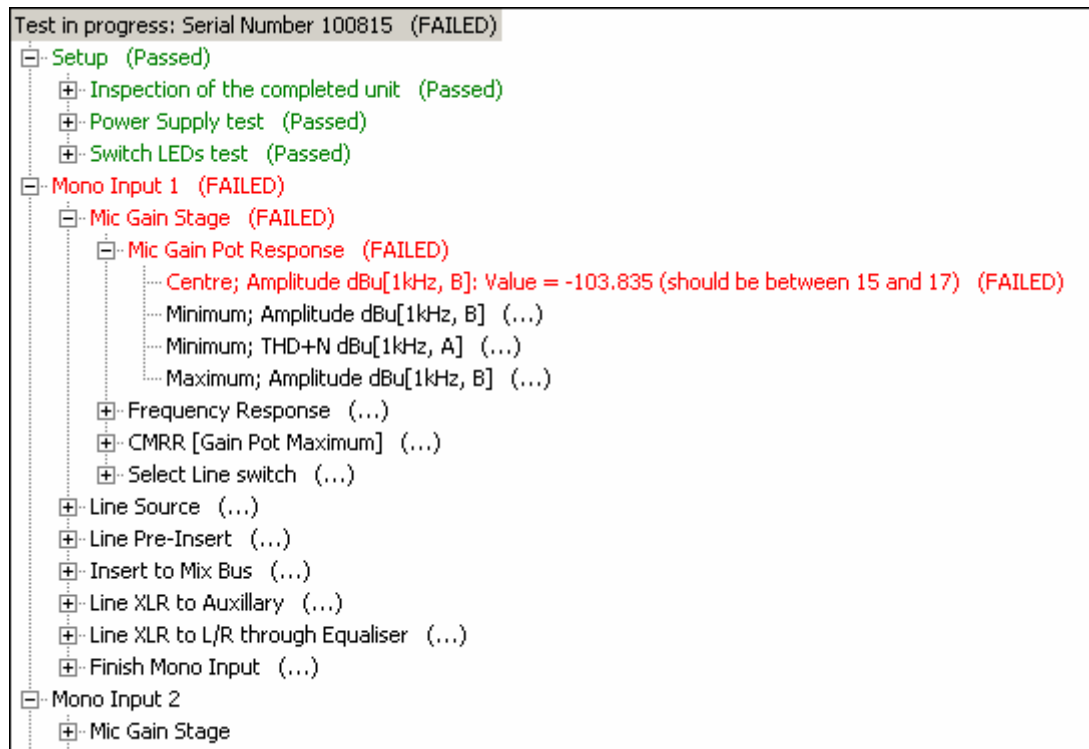
Product Family	Allows selection of the Product Family of the Product that will be tested. By default, this will be the last Product Family used. When the Product Family is changed, the Product list (see below) will be refilled with the Products in the new family.
Product	Allows selection of the Product to be tested. By default, this will be the last Product used. When a new Product is selected, the Tree of results will be updated to reflect the structure of tests to be done on this Product.
Operator	Shows the current Operator logged in to AutoTestSQL. Use the Change button to log in as a different Operator (or the Change Operator option on the Utility menu).
Change >>	Allows the current Operator to be changed. This means that a different Operator can log in to perform a test, without having to exit from the AutoTestSQL application.
Run	Runs a test. This will run the various Scriptlets for this Product, log the results to the database, and display the results in the Tree of results on the right-hand side.
Stop	Stops the currently running test.
When the current test fails...	Allows selection of the action to take when a failed result is detected. This can be to continue with the test, stop the test, or ask the Operator whether they would like to re-test this Test Set. See Re-testing failures for further details.
Show ... levels	This determines how many levels to show in the Tree of results . Since there will usually be a lot of hierarchically-stored results in the tree, it is more likely that individual branches of the tree will be opened and closed when necessary. Note that it may take some time to open the Tree of results , if there are a lot of Test Sets at the required level.



Some of these fields can be enabled or disabled depending on whether a test is currently running. For example, the [Stop] button is disabled unless a test is running, and the [Run] button is disabled whilst a test is running.

6.4.2 Tree of results

In Test mode, the right-hand panel of the screen contains the hierarchical tree of tests and results for the currently selected Product:



This tree initially contains the tree of Test Sets and Test Elements for the selected Product. As the test is run, corresponding Result Sets and Result Elements will be created in the database. When this happens, the display will be updated with the status of the results.

Note that when a [Scriptlet](#) is run, all the Result Sets and Result Elements below it in the hierarchical structure will be created. When this happens, all Result Sets and Result Elements will be shown with a status of (...) appended. This means an unknown status, i.e. the Result exists in the database but has not yet been marked as a pass or a failure.

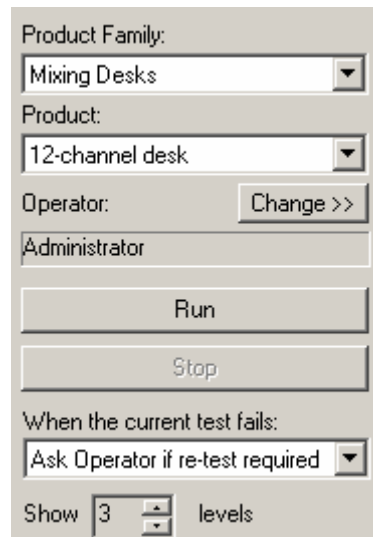
As the Scriptlet runs, the status of these results will be updated in the tree of results. A [failed result](#) will be shown in red, with (FAILED) appended to it. A [passed result](#) will be shown in green, with (Passed) appended. Any [non-critical results](#) will be updated with the value of the result; the (...) will be removed and the display colour will not be changed.

For full details on how Product and Test Scriptlets are used to run tests, and how this affects when the results are created in the database, see [Test Scriptlets](#).

6.4.3 Running a test

Before running a test, you will need to have at least one Product set up in AutoTestSQL with a set of tests defined for it. To find out how to set up tests, see [Setup mode](#).

To run a test, you will need to be in Test mode. Once in Test mode, you will see the following options in the left panel of the AutoTestSQL screen:



The screenshot shows a control panel with the following elements:

- Product Family:** A dropdown menu with "Mixing Desks" selected.
- Product:** A dropdown menu with "12-channel desk" selected.
- Operator:** A text field containing "Administrator" and a "Change >>" button.
- Run:** A large button to start the test.
- Stop:** A large button to stop the test.
- When the current test fails:** A dropdown menu with "Ask Operator if re-test required" selected.
- Show:** A numeric spinner set to "3" followed by the text "levels".

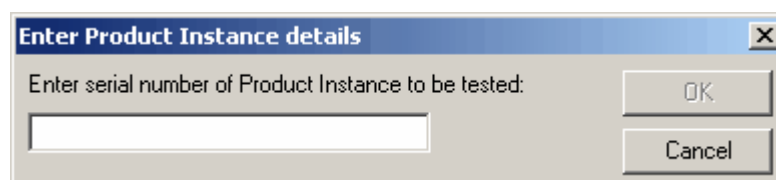
You will firstly need to ensure that you are the currently logged-in Operator. This information will be displayed under "Operator". If this is a different Operator, select the [Change] button to log in as yourself. This will bring up the [Login dialogue box](#). Making sure you are logged in correctly will ensure that the correct Operator details are recorded in the database for this test.

Once you are correctly logged in as the current Operator, you must select the Product that you wish to test. To do this, firstly select the correct Product Family from the top drop-list, and then the Product from the second drop-list. When you do this, the test structure for this Product will be created and the list of results on the right-hand side of the screen will be updated.

Once the test structure has been created, specify the action you wish to take when a [failed result](#) occurs, and then click the [Run] button to start the test.

[Entering Product Instance details](#)

When the test starts, you will be prompted to enter the serial number of the device that you are testing:



The dialog box has a title bar "Enter Product Instance details" with a close button (X). Inside, it says "Enter serial number of Product Instance to be tested:" followed by a text input field. At the bottom right are "OK" and "Cancel" buttons.

You can define your own validation for this serial number in your script - see the description of the [ValidateSerialNumber](#) function for details.

If you have elected to "Stop the test" when a [failed result](#) is detected, then the script will stop running when the first failed result is logged in the database. Otherwise, the test will run through to completion, or will run until the [Stop] button is clicked. See [Stopping a test](#) for further details.

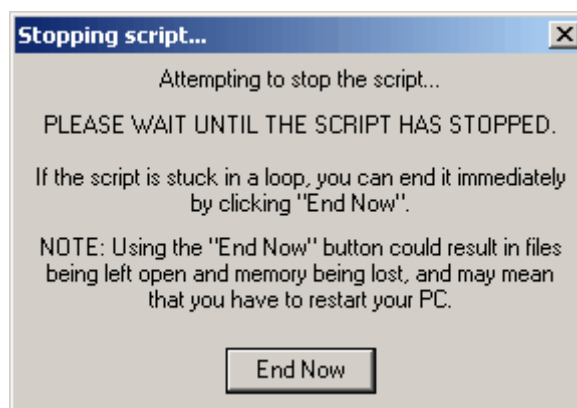


The default selections for the Product Family and Product will be the last ones used. This information is remembered between AutoTestSQL sessions, as is the action to take on test failure.

6.4.4 Stopping a test

In Test mode, you can stop a test at any point by clicking the [Stop] button.

When the script is stopped in this way, the AutoTestSQL program must ask the script to stop at the next convenient point. It is possible that it may be able to stop immediately, although more likely that there will be some delay in this process. While the script is trying to stop, the following message will be displayed:



The most likely place for the script to be able to stop is the next point at which a call is made to the AutoTestSQL software using one of the built-in [Script Functions](#). For example, the script will contain many calls to SetResult to write its details to the database; After clicking the [Stop] button, it is likely that the script will reach another SetResult call very soon and can end neatly.

Under some circumstances, the script may never be able to stop at a convenient point - for example, if the script is stuck in an endless loop due to a bug in the script (See [Common script problems](#)). In this case the above message will remain on the screen, and you will only be able to stop the script using the [End Now] button. It is recommended that the [End Now] option is not used unless absolutely necessary, since there is a risk that it may leave the script without tidying up properly; for example without cleaning up memory, or leaving the dScope object open.

For details of what happens when the script is stopped, see [Test completion](#).

6.4.5 Re-testing failures

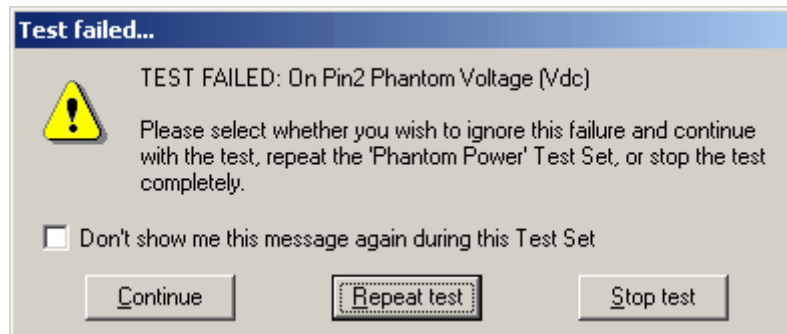
In Test mode, you can specify the action that you wish to take when a [failed result](#) is detected during a test.

There are three options available:

Continue with test: This option simply ignores the fact that a failure has occurred, and continues with the test.

Stop the test: This option stops the test immediately when a failed result is detected. The current Test Set Scriptlet is stopped, and the test is finished neatly by calling the Product Scriptlet's [OnTestFinish](#) function, if present.

Ask Operator if re-test required: When the failed result is detected, the Operator is shown the following options:



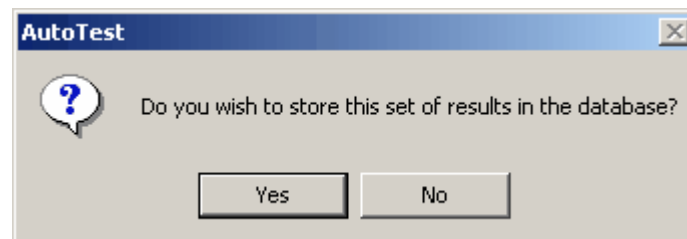
Selecting "Continue" will simply carry on with the test. "Repeat test" will stop the currently running Test Set Scriptlet, reset the status of all Result Elements within the Test Set to "Unknown", and restart the Scriptlet. "Stop Test" will simply stop the test at the current position, calling the Product Scriptlet's [OnTestFinish](#) function to finish neatly.

6.4.6 Test completion

When a test is stopped, either by clicking the [Stop] button while the test is running, or automatically due to completion of the test or stopping due to a failure, a number of things may occur. These will happen according to the selections made in the [Options dialogue box](#) - see this section for full details of the options available.

[Discarding results](#)

An option can be set to ask the Operator whether to save or discard the results at the end of the test. If this option has been turned on, you will see the following message:

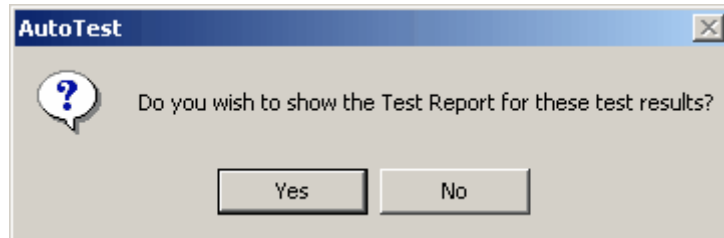


This option is useful when setting up and debugging tests, to ensure that the database does not get filled with debugging results that are not required. Select [Yes] to write the results to the database, or [No] to discard the results.

Note that in order to allow the Operator to carry on using AutoTestSQL (for example, to enter Fault-find mode), the results are actually saved quickly to a temporary file. A background [thread](#) will then write the results to the database in its own time, without interrupting any actions the Operator may be performing.

Test reports

AutoTestSQL can be set up to Show, Print and/or Save details of the test in a Test Report. Any combination of these options can be set, together with whether the Operator should be asked at the end of the test, or whether the report generation should occur automatically. If this option has been set, then at the end of the test, you will be asked whether you wish the Test Report to be created:



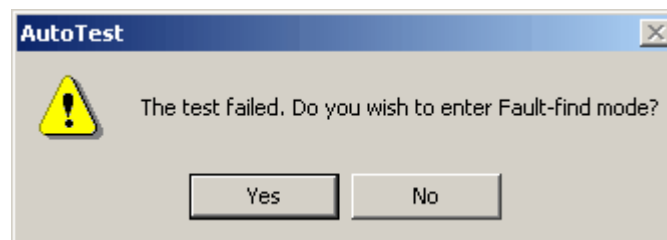
NB:

The text in this message will change depending on which combination of show, save and print have been selected in the Options dialogue box.

If you select [Yes] then the Test Report will be created. This may take several seconds depending on the size of the test structure. For a full description of the test report, see [The Test Report](#).

Fault-find mode

If the test failed, you will be given the option to enter [Fault-find mode](#). This gives you the opportunity to recreate the failure situation, rectify it, and log details to the database:



Clicking [Yes] here will cause the current mode to change to Fault-find mode, and the Test Session just completed will be listed in the tree of results on the right-hand side.

6.4.7 The Test Report

A Test Report can be created once a test has finished, showing full details of all the results for the test session. The test report is created as an [HTML](#) file. Depending on the settings for the Test Report in the [Options dialogue box](#), this will either be saved as a temporary file or saved permanently according to the filename structure specified. When the report is displayed or printed, it will have the following format:



The Test Report simply lists all the results in the test session, together with their status, and formats them using a .css ([cascadable style sheet](#)) file (see below).

[The Cascadable Style Sheet File](#)

The Cascadable Style Sheet contains formatting details for the Test Report. It is stored with a file name of **TestReport.css** in the Test Reports [subfolder](#) of the AutoTestSQL [program folder](#). It contains formatting details of each type of entry in the report - Result Sets, Result Elements, etc. and also for each result status (Passed, Failed, etc.). This file can be changed if required to alter the format of the report.

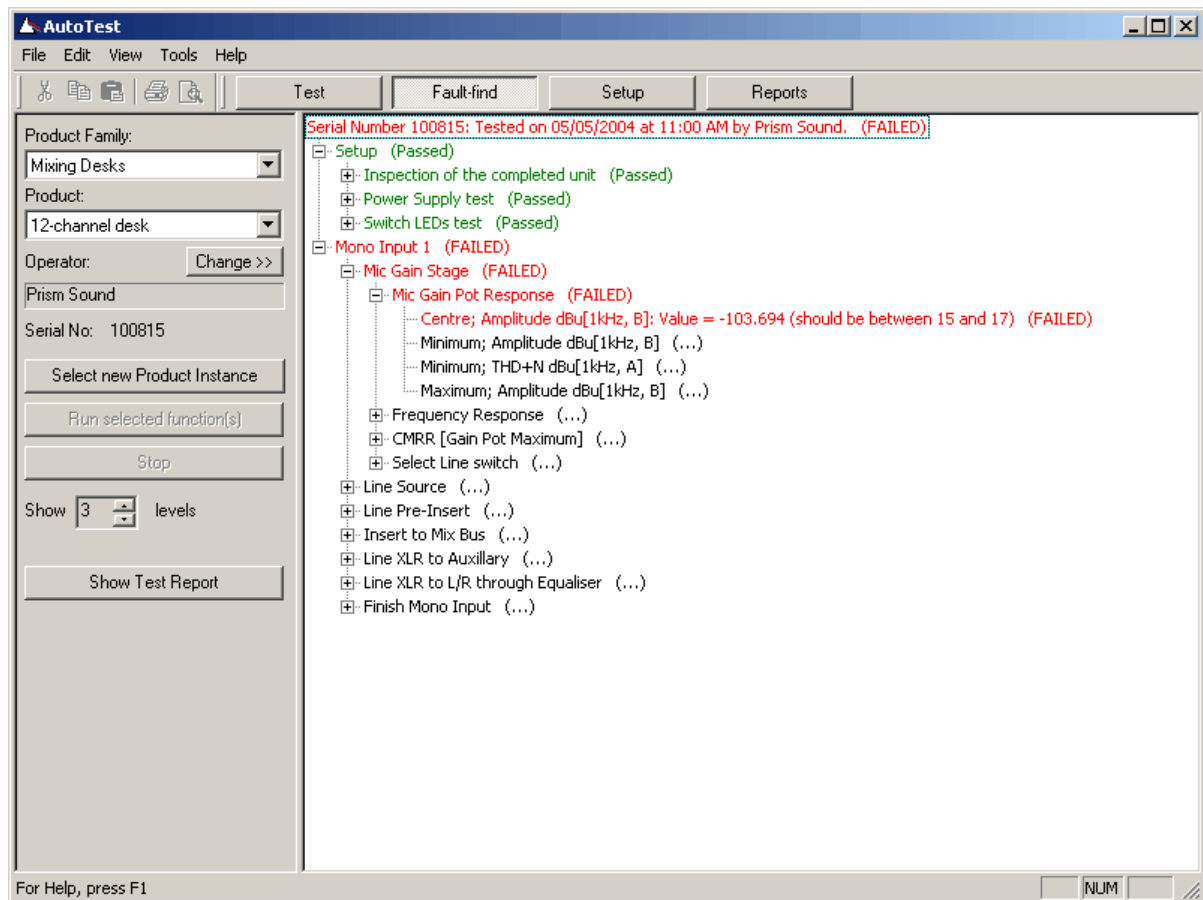


The location of the Test Reports folder is defined in the [Options dialogue box](#).

6.5 Fault-find mode

Fault-find mode allows you to investigate a failure in a test, and determine its cause. It allows you to select an individual [failed result](#), and then get the test equipment back to exactly the same condition as it was in when the failure occurred. In this way the available equipment (for example, the dScope) can be used to investigate the cause of failure before logging details of the fault to the database. The fault can be rectified at this point for the Product Instance under test, and since the details have been stored in the database, future analysis can be done of the most common reasons for failure of a test.

The screen in Fault-find mode is similar to that of Test mode. It consists of a series of fields on the left hand side, and the hierarchical tree of results on the right. This tree of results is identical to the tree of results created in Test mode, and allows the Operator to select a failed result to investigate.



The following sections describe the operation of Fault-find mode:

[Fields and controls](#)
[Selecting a failed result](#)
[Entering Fault details](#)

6.5.1 How Fault-find mode works

Fault-find mode works by getting the test equipment back to exactly the same situation that it was in when a failure occurred. This enables any available equipment (for example, the dScope, or a multi-meter) to be used to investigate the cause of failure, since the [EUT](#), the dScope, and any other equipment should be in exactly the same state as it was when the fault occurred.

To do this, AutoTestSQL runs selected Test Set scriptlets. It is usually not acceptable to run all the Scriptlets up to the point of failure, as this could take a long time. However, it must run enough of the Scriptlets to ensure that the equipment is in the same state - for example, the correct dScope Configuration must have been loaded; any automated control of the EUT must have occurred (for example, downloading of patches); any switching of channels using an [I/O Switcher](#) must have been done; and the correct inputs and outputs of the EUT must be selected in the same way as when testing.

In Fault-find mode, the Operator selects the Result Element under investigation in the tree of results. AutoTestSQL will search from this item upwards through its parent Result Sets until it finds a [Scriptlet](#). This Scriptlet will be run in an attempt to get the system back to the state of failure. If more than one Result Element is selected, this process will be applied to each of them in turn, creating a list of the Scriptlets to be run.

Usually, the Operator will select a single Result Element, or a Result Element together with all its parent Result Sets up to the top-level Result Set in the Product's test session (see [Selecting a failed result](#)). In the latter case, AutoTestSQL will create a list of the Scriptlets to be run, starting with the top-level Result Set, and working its way down to the Result Element. This ensures that the Scriptlets are run in the same order that they were run when the test was performed.

The [OnTestStart](#) and [OnTestFinish](#) functions in the Product Scriptlet will also be run during Fault-find mode, as in Test mode.



In Fault-find mode, no results are written to the database.

6.5.2 Fields and controls

In Fault-find mode, the left-hand side of the screen contains the various fields and controls available. These are similar to the controls found in Test mode:

The screenshot shows a control panel with the following elements:

- Product Family:** A dropdown menu showing "Mixing Desks".
- Product:** A dropdown menu showing "12-channel desk".
- Operator:** A text field containing "Prism Sound" and a "Change >>" button.
- Serial No:** A text field containing "100815".
- Buttons:** "Select new Product Instance", "Run selected function(s)", "Stop", and "Show Test Report".
- Show:** A numeric spinner set to "3" followed by the text "levels".

The following is a list of the available fields and controls, and their function:

Product Family	Allows selection of the Product Family of the Product whose fault is to be investigated. By default, this will be the last Product Family used. When the Product Family is changed, the drop-list of Products (see below) will be refilled with the Products in the new family.
Product	Allows selection of the Product whose fault is to be investigated. By default, this will be the last Product used. When a new Product is selected, the test structure for this Product will be recreated and the tree of results will be updated to reflect the tests to be done on this Product.
Operator	Shows the current Operator logged in to AutoTestSQL. Use the Change button to log in as a different Operator (or the Change Operator option on the Utility menu).
Change >>	Allows the current Operator to be changed. This means that a different Operator can log in to perform fault-finding, without having to exit from the AutoTestSQL application.
Serial No.	Shows the serial number of the Product Instance currently under investigation.
Select new Product Instance	Use this button to select a new Product Instance to perform fault-finding for. See Selecting a new Product Instance (below) for further details.
Run selected function(s)	Runs the parts of the script relevant to the currently selected Result Element in the tree. See Selecting a failed result for full details.
Stop	Stops the currently running fault-find script.
Show ... levels	This determines how many levels to show in the tree of results . Since there will usually be a lot of hierarchically-stored results in the tree, it is more likely that individual branches of the tree will be opened and closed when necessary. Note that it may take some time to open the tree of results, if there are a lot of Test Sets at the required level.
Show Test Report	Recreates and shows the Test Report for this set of results. See The Test Report for more details.

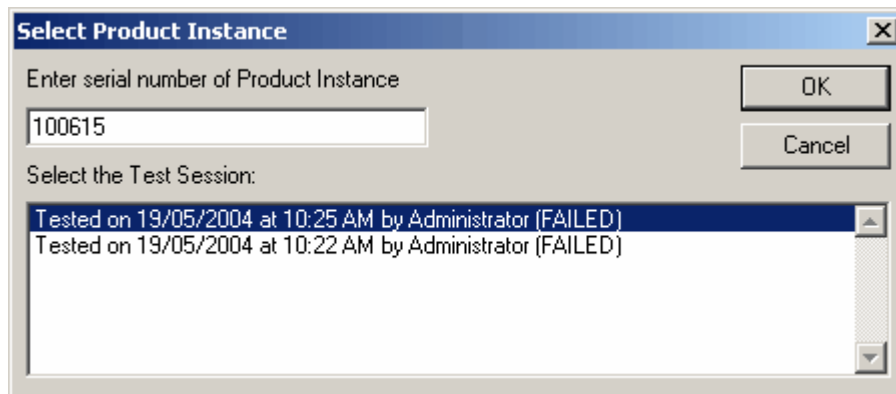


Some of these fields can be enabled or disabled depending on whether the script is currently running. For example, the [Stop] button is disabled unless a the script is running, and the [Run] button is disabled whilst the script is running.

[Selecting a new Product Instance](#)

Whilst in Fault-find mode, you may wish to investigate a fault from a different Test Session, or indeed a different Product entirely. For a Test Session for a different Product, you will need to firstly select the Product from the Product drop-list.

To select a new Test Session, click the [Select New Product Instance] button. The following dialogue box will open:



Enter the Serial number of the Product that you wish to choose. When you exit from the Serial number field, the list at the bottom will be filled with all the Test Sessions for this product, showing the date, time, Operator and status of the Test Session. If the [Options dialogue box](#) has been set up to allow the same serial number to be re-used for different Products, then this list will be limited to the instances of the currently selected Product in the drop-list on the left-hand panel of the Fault-find screen.

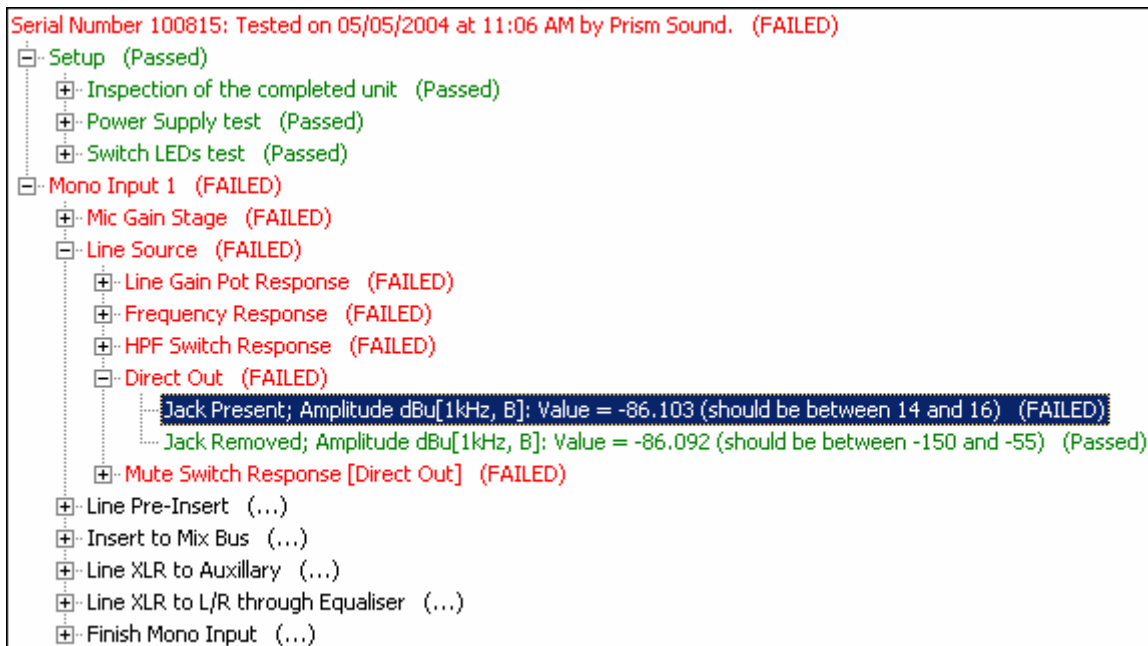
To choose one of the Test Sessions listed, click the [OK] button. The dialogue box will close, and the list of results on the Fault-find screen will be filled with the details of the selected test session. Alternatively, click [Cancel] to return to Fault-find mode without changing the current Test Session.

6.5.3 Selecting a failed result

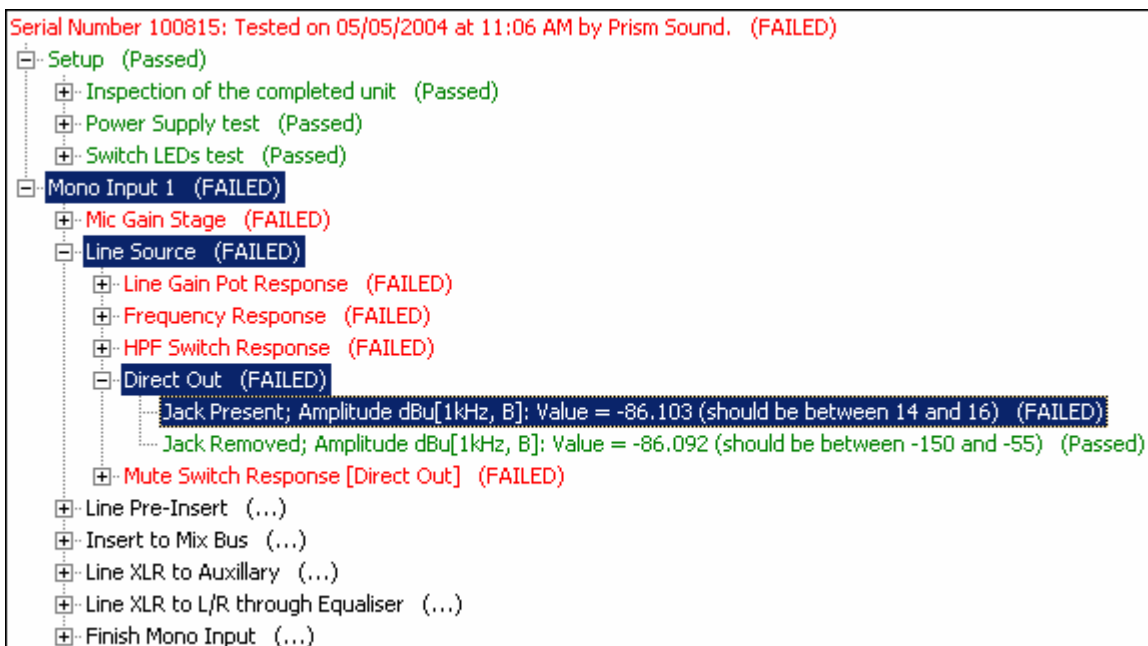
In Fault-find mode, you will select a result from the list of results in the right-hand panel of the AutoTestSQL screen. AutoTestSQL will then attempt to run the scripts that were run in [Test mode](#) to get to the situation where this result failed.

For a description of the operation of Fault-find mode, see [How Fault-find mode works](#).

There are two ways of selecting a failed result - you can either select the individual result, as shown below:



Alternatively, by holding down the <Alt> key whilst clicking, you can select the result and all its parents in the tree.



Note that there is an important difference between these two selection methods, and it may require some knowledge of the structure of the tests in order to know which to choose. See [Knowledge of the test structure](#) for further details.

Fault-find mode will search up the Result tree and run the first Scriptlet that it finds to attempt to get to the fault situation. If a single Result Element is selected, then only one Scriptlet will be run; if the Result Element and all its parent Result Sets are selected, the Scriptlet for each one (if present) will

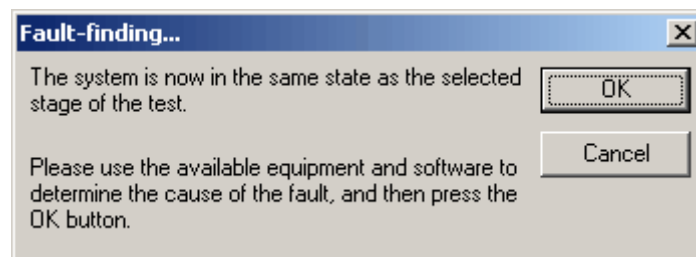
be run when getting AutoTestSQL back into the fault condition. See [How Fault-find mode works](#) for a full description of this process.



Note that the [Run selected function(s)] button will be disabled until a Result Element is selected.

6.5.4 Entering fault details

In Fault-find mode, clicking the [Run selected function(s)] button will cause the relevant Scriptlets to be run up to the point where the failure occurred. When this point is reached, the following message will be displayed:



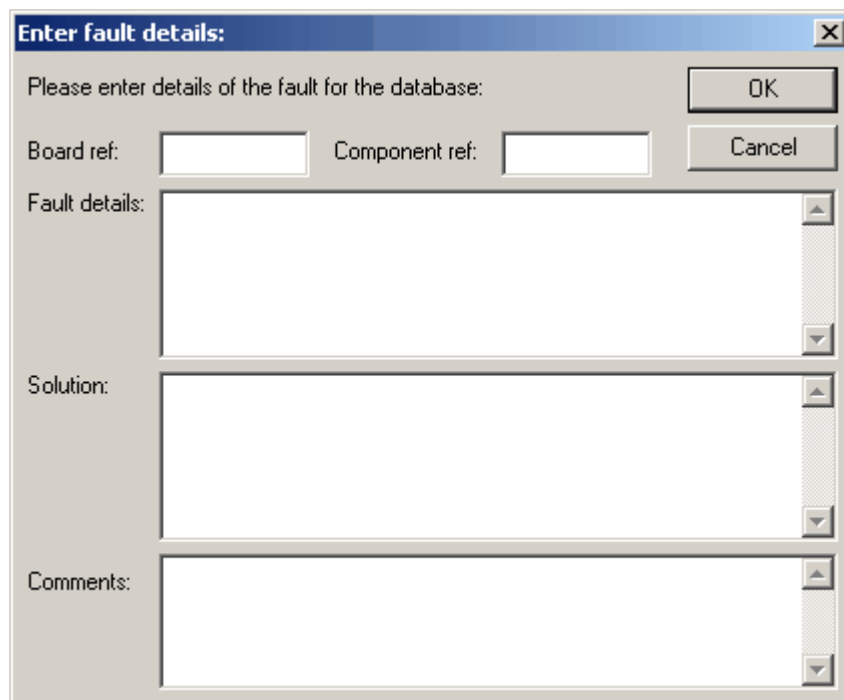
The system should now be in the same state as it was in Test mode when the failure occurred. You can now use all available equipment (multi-meters, the dScope software, etc) to work out where the problem lies and, if necessary, correct it.



Use the <Alt> + <Tab> keys together to cycle through the currently open Windows applications, and bring the dScope to the front of the screen.

Entering details of the fault

When you have found the problem, click [OK] to let you enter details of the fault in the database, or [Cancel] to exit without logging the fault details. Clicking [OK] will bring up the following dialogue box:

A Windows-style dialog box titled "Enter fault details:". It contains a label "Please enter details of the fault for the database:" followed by two text input fields: "Board ref:" and "Component ref:". Below these are three larger text areas: "Fault details:", "Solution:", and "Comments:". Each text area has a vertical scrollbar on its right side. In the top right corner of the dialog, there are two buttons: "OK" and "Cancel".

The following details can be entered into the database:

Board ref.	Use this field to enter a reference to the board on which the component causing the fault is located. This can be up to 20 characters long.
Component ref.	Enter the reference of the component causing the fault. This can be up to 20 characters long.
Fault details	Use this field to enter details of the fault. This can be up to 1000 characters.
Solution	Use this field to enter details of how the fault was corrected. This can be up to 1000 characters.
Comments	Use this field to enter any additional comments about this fault. This can be up to 1000 characters.
OK	Exits from this dialogue box and saves the fault details in the database.
Cancel	Exits from this dialogue box without saving any fault details to the database.

These fault details can then be used for [management reporting](#) purposes, to enable easy analysis of where the most faults occur.

6.5.5 Knowledge of the test structure

Ensuring that Fault-find mode operates correctly may require some knowledge of the order in which the various test [Scriptlets](#) are integrated into the test structure. For full details of how the test structure is put together, see the sections on [Test structure](#) and [Test Scriptlets](#).

For example: consider the following Test Set structure, the associated Scriptlets, and the tasks that each Scriptlet performs:

```

Mono Inputs (Scriptlet: MonoInputs.vbs)
  Input 1 (Scriptlet: MonoInputN.vbs)
    Mic Gain Stage (No Scriptlet)
      Mic Pot Response (Scriptlet: MicPotResponse.vbs)
        Centre
          Amplitude (dBu)
          THD+N (%)
        Minimum
          Amplitude (dBu)
          THD+N (%)
        Maximum
          Amplitude (dBu)
          THD+N (%)
      Input 2 (Scriptlet: MonoInputN.vbs)
        (As for Input 1)
      ..
      Input N (Scriptlet: MonoInputN.vbs)
        (As for Input 1)

```

Assume for this example that the Scriptlets associated with these tests do the following:

MonoInputs.vbs	Loads a dScope Configuration .
MonoInputN.vbs	Asks the Operator to plug the Mono Input into the correct channel of the EUT, and to set all pots for the channel to their mid-point.
MicPotResponse.vbs	Displays messages asking the Operator to set the Pot to Centre, Minimum and Maximum; for each of these, it measures the Amplitude and THD+N using the dScope.

If one of these Amplitude or THD+N measurements failed, then you would use Fault-find mode to work out why. You would have two possible options for running the scripts: selecting *only* the failed Result Element, or selecting the Result Element and all its parents (See [Selecting a failed result](#) for full details of how to do this).

Selecting only the Result Element would cause AutoTestSQL to search up the tree until it found the "MicPotResponse" Scriptlet. This Scriptlet would be run, taking the Amplitude and THD+N measurements.

However since no dScope Configuration has been loaded at this point, and the Operator has not been asked to make the required connections to the EUT, both the dScope and the EUT are in an undefined state and so the results may not correspond to those made during the test.

In this example, the correct option would be to select the Result Element *and* its parents. This would search up the tree and run the MonoInputs, MonoInputN and MicPotResponse Scriptlets in that order. This would ensure that the relevant dScope Configuration was loaded; the Operator would be shown a message asking for the correct connections to be made to the EUT; and the Amplitudes and THD+N measurements would be made under the same circumstances as they were during the test.

As you can see, in this example, some knowledge of what the test's Scriptlets contain is necessary for correct operation in Fault-find mode.

6.6 Reports

Management reporting will be managed by an external report designer, such as Crystal Reports.

The application to be used for this can be specified in the Options dialogue box; this application is then invoked when the AutoTestSQL Toolbar's Reports button is clicked.

For a full description of management reporting with AutoTestSQL, see [Management reporting](#).

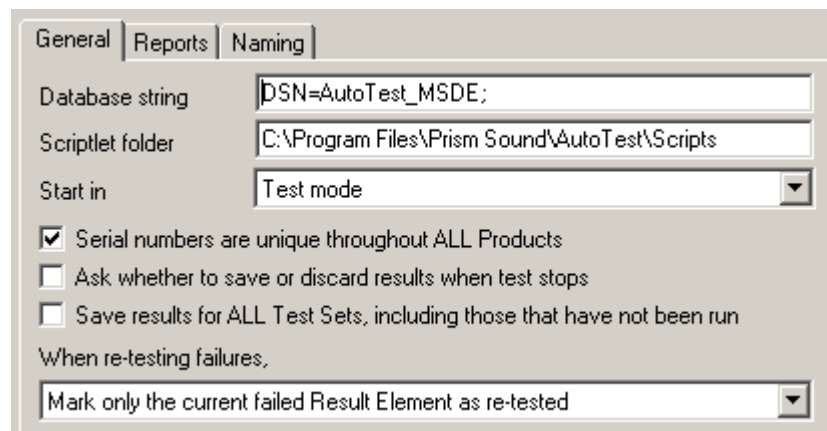
6.7 Options dialogue box

The Options dialogue box is available from AutoTestSQL's Utility menu. It allows the Operator to specify a variety of miscellaneous operating options for AutoTestSQL. These are stored in the Windows [registry](#), and are thus retained for all future sessions.

The Options dialogue box is split into three parts, each one with its own tabbed section:

[General tab](#)

The General tab stores miscellaneous AutoTestSQL options:



Database string: This field contains the string used to connect to the database. By default this contains a string used to open an [MSDE](#) database on the local machine, i.e:
`DSN=AutoTest_MSDE;`



Since this is accessing the same database as the [AutoTestSQL Report Viewer](#) software, the Database string is stored in the same place in the registry for both AutoTestSQL and the Report Viewer. Changing this value will also affect the Report Viewer software's database opening details.

Scriptlet folder: This field defines the base folder for all [Scriptlets](#). In Setup mode, if Test Set Scriptlets are specified as relative paths, they are assumed to be relative to this base folder. If this folder name contains spaces, it must be enclosed in double-quotes. On installation, this entry is set by default to <AutoTestSQL program folder>\Scripts, where <AutoTestSQL program folder> is the [folder](#) to which the AutoTestSQL program has been installed.

Start in: This allows you to specify which mode you would like AutoTestSQL to start in. This can be set to Test mode, Fault-find mode, Setup mode, or the last mode that you were in when you finished using AutoTestSQL. The default is "Test mode".

Serial numbers are unique throughout all products: Checking this option means that you cannot use the same serial number for different Products. If unchecked, then the same serial number can be reused for different Products, even within different Product Families. This option defaults to checked.

Ask whether to save or discard results when test stops: If this box is checked, then on completion of a test, the Operator will be asked if they wish to store the test results in the database. If they select No, then any results created as part of the test will be deleted from the database. This can be particularly useful when setting up and debugging tests, as you may not want to store the results in the database until you are happy with your test setup. This option defaults to checked.

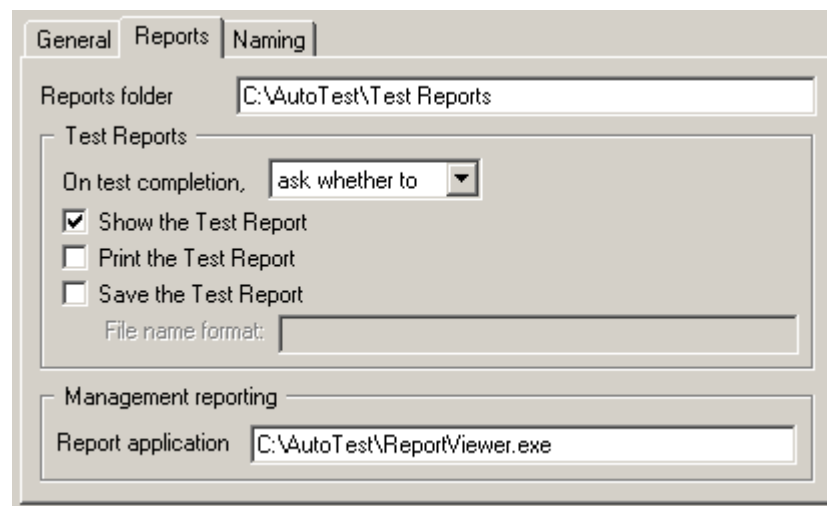
Save results for ALL Test Sets...: A test may be stopped part way through for one of a number of reasons, which may leave some of the Test Sets "Untested". This option can be used to specify that *all* results, even those untested, are written to the database. If left unchecked, then only Test Sets that have been tested (either partially or completely) will be saved.

When re-testing failures, ...: When a [failed result](#) is detected, the Operator can re-test the Test Set. This drop-list can be used to specify whether you wish to make a note of whether the Result Elements within the current Result Set have been re-tested. The following options are available:

Mark all Result Elements as re-tested	When the Test Set Scriptlet runs to set result values again, <i>all</i> Result Elements that have already been tested are marked as re-tested.
Mark only the current failed Result Element as re-tested	Only the current Result Element, the one causing the failure, is marked as re-tested.
Mark all failed Result Elements within this Result Set as re-tested	Any Result Elements that have a "Failed" status are marked as being re-tested (any that are Passed, Unknown or non-critical are not).
Do not mark any Result Elements as re-tested	This option ignores whether an item has been tested more than once, and simply marks it as Passed or Failed.

[Reports tab](#)

The Reports tab contains options pertinent to [Test Reports](#) and [Management reporting](#) in AutoTestSQL.



Reports folder: This field contains the location of the base folder for Test Reports. All Test Reports are stored in this [folder](#) or its [subfolders](#). If this folder name contains spaces, it must be enclosed in double-quotes.

On Test completion, ... : This options allows you to specify whether the selected options for Test Reports should always be applied, or whether the Operator should be asked. This option defaults to "ask whether to..."

NB: If the subsequent Show, Print and Save Test Report fields are left unchecked, no Test Report will be created and the selection in this field is ignored.

Show the Test Report: If checked, then the Test Report will be shown to the Operator on test completion. This option defaults to checked.

Print the Test Report: If checked, then the Test Report will be printed on test completion. This option defaults to unchecked.

Save the Test Report: If checked, then the Test Report will be saved to the hard disk on test completion. This option defaults to unchecked.

File name format: This specifies the format of the name of the Test Report file to be saved to the hard disk, if the preceding **Save the Test Report** check box is checked. Test Reports will be saved within the Test Reports folder specified above, using this format. This field can substitute various items of text from the current Test Session to ensure uniqueness of file names. This field is blank by default.

The following items can be substituted into the file name format:

[ProductFamily]	The name of the Product Family.
[Product]	The name of the Product.
[Operator]	The name of the currently logged-in Operator.
[OperatorID]	The ID number of the currently logged-in Operator.
[SerialNum]	The Serial number of the Product Instance being tested.
[Year]	The current year, as a four-digit number.
[ShortYear]	The last two digits of the current year.
[Month]	The current month, as a two-digit number (01 to 12)
[Day]	The current day of the month, as a two-digit number (01 to 31)
[Hour24]	The hour that the test was started, in 24-hour format (00 to 23)
[Hour12]	The hour that the test was started, in 12-hour format (01 to 12)
[Minute]	The minute that the test was started (00 to 59)
[AutoInc]	Whether to auto-increment the file name if the file name already exists. If it does, then the [AutoInc] text in the file name will be replaced by the next number in sequence, starting at 1. If [AutoInc] is not specified in the file name format, then whenever AutoTestSQL tries to save a Test Report with a file name that already exists, the Operator will be asked to either confirm overwriting the file, or to specify a new file name.

For example, entering a file name format of:

[ProductFamily]\[Product]\[SerialNum]_[Year][Month][Day]_[Hour24][Minute].html

will ensure that each Test Report will be stored in a folder with the Product Family name, within a subfolder with the Product name, and with a file name consisting of the serial number, the date and the time of the test.



If the file name resulting from applying this format contains invalid filename characters, these will be removed from the string before the file is saved.

Report application: This contains the full path name of the application used to produce management reports. This application will be run when the "Reports" button is clicked from the main toolbar. If this application name contains spaces, it must be enclosed in double-quotes.

Naming tab

AutoTestSQL allows you to use your own names for several of its key areas. This part of the Options dialogue box allows you to specify the names you want to give to each item:

Entity	Default Name
Product Family	Product Family
Product	Product
Product Instance	Product Instance
Test Set	Test Set
Test Element	Test Element
Operator	Operator
Scriptlet	Scriptlet
Result Set	Result Set
Result Element	Result Element

Product Family: A group of Products.

Product: An item that will be tested.

Product Instance: Individual instances of Products. A Product Instance equates to a single item of Equipment Under Test (EUT).

Operator: A user of AutoTestSQL, that might set up the tests, or perform testing and fault-finding.

Scriptlet: An individual script file that is associated with a Test Set.

Test Set: A group of tests. This group may contain Test Elements, and/or other Test Sets.

Test Element: A Test entity that corresponds to a single value (numerical, pass/fail or text) that will be entered into the database.

Result Set: A group of results. This group may contain Result Elements, or other Result Sets.

Result Element: A single result value that will be entered into the database.

6.8 AutoTestSQL security

AutoTestSQL contains a basic security system that allows different Operators to be given different levels of responsibility. There are a number of operations that each Operator can be allowed to perform, or denied access to. For example, accessing the AutoTestSQL Options using the [Options dialogue box](#) may be an activity that is restricted to all but the most advanced Operators.

Each Operator on the system is given a password, and must use this password to login to AutoTestSQL. This ensures that Operators can only log in as themselves and not as anyone else. Once using the system, the Operator may be denied access to certain parts of AutoTestSQL depending on the operations that they have been given access to when their Operator details were entered (See [Entering/editing Operator details](#)).

The Administrator Operator

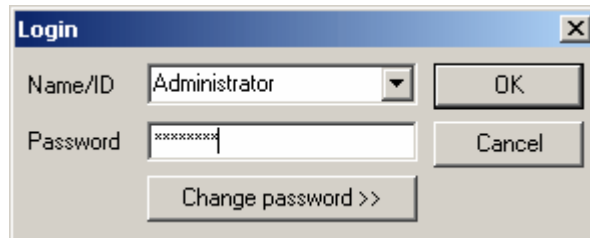
AutoTestSQL contains a built-in Administrator Operator that cannot be removed (although it can be renamed, if necessary). This Administrator has full access to all parts of the system, and can therefore be used to set up and edit details of other Operators on the system.

NB: When AutoTestSQL is first run, this Administrator is the only Operator in the database and you will be required to log in as the Administrator (see below) before accessing AutoTestSQL. The

password is initially blank, and you will be requested to change it before continuing.

The Login dialogue box

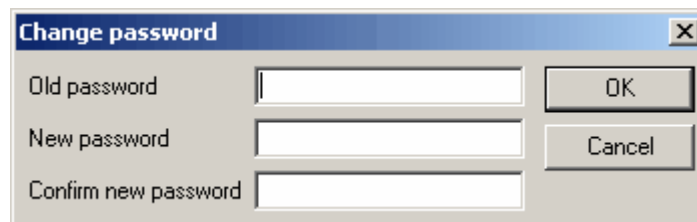
The Login dialogue box is always displayed when AutoTestSQL is run. It can also be accessed by clicking the [Change >>] button in Test or Fault-find mode, or using the Change Operator option on the Utility menu.



Name/ID	Select the name of the Operator to log in as from the drop-list, or type in the Operator ID or name.
Password	Enter the password of the Operator to log in as. This password will be displayed as asterisks to preserve secrecy.
Change password >>	Click this button to change the password of the Operator currently selected in the Name/ID drop-list. See Changing passwords for details of this option.
OK	Attempts to log in as the specified Operator. If the name or password is incorrect, you will be informed and asked to try again; otherwise, the dialogue box will be closed and you will be logged in with these Operator details.
Cancel	Closes the Login dialogue box without attempting to log in as the current operator. If this dialog box is being displayed on startup, the AutoTestSQL application will close. Otherwise, the previous Operator will stay logged in.

Changing passwords

When setting up Operators, it can be specified that the Operator must change their password the next time they log in to AutoTestSQL. It is also possible for an Operator to change their password by clicking on the [Change password >>] button on the Login dialogue box. In either situation, the following dialogue box will be displayed:



Old password	For security reasons, the old password must be entered before it can be changed. Entries made in this field will show as asterisks.
New password	The new password to change to. Entries made in this field will show as asterisks.
Confirm new password	Re-enter the new password as confirmation. Entries made in this field will show as asterisks.
OK	Providing the old password is correct, and the new password is valid and is correctly confirmed, this will close the dialogue box and a confirmation message will be given that the password has been changed. Otherwise, the Operator will be informed of the failure and asked to try again.
Cancel	Closes the dialogue box without changing the password.

Part



7

The AutoTestSQL database

7 The AutoTestSQL database

This section of the manual contains a detailed description of the AutoTestSQL database. It is not necessary to read or understand this section in order to use AutoTestSQL; however a fuller understanding may help when designing a test structure and writing Scriptlets. It will be necessary to have some knowledge of the database structure to produce [management reports](#).

For a description of how the database is structured, or the fields in each table, see [Database structure](#) or [Database tables](#).

To find out how to write your own SQL commands to query the AutoTestSQL database, see [Using SQL from the command-line](#).

For important information about backing up and restoring the database, see [Backing up and restoring the database](#).

To find out how to set up a multi-user version of AutoTestSQL, see [Multi-user setup](#).

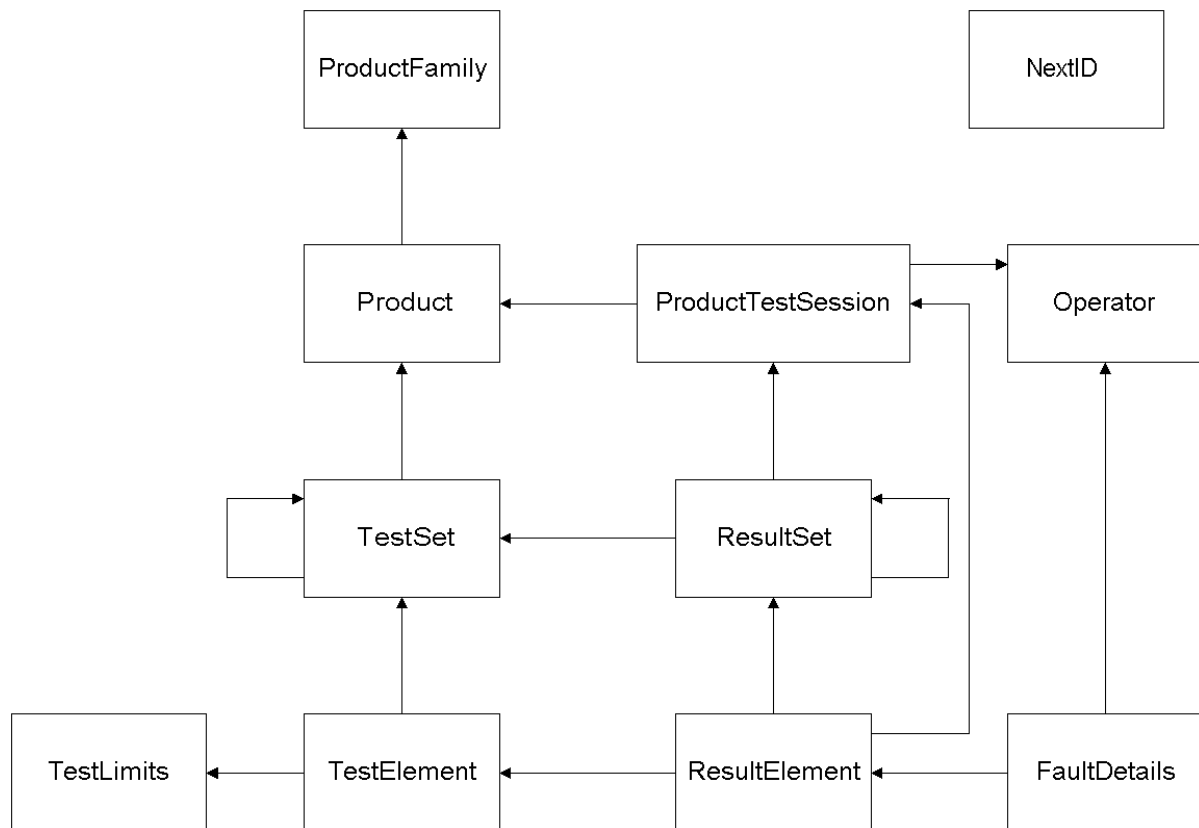
For more information on the AutoTestSQL software's database [MSDE](#) and its relationship to SQL Server, see [Limitations of MSDE](#).

7.1 Database structure

An AutoTestSQL database stores its data in [tables](#). A table represents an entity in the AutoTestSQL structure; for example a Product, an Operator or a Result Set.

Each table contains a row for each record of the specified type. The columns in the table represent the different fields of the record - for example a Product table contains a column for the Product's name, and a column for the Product's parent Product Family.

The following diagram shows which tables in the AutoTestSQL database are linked together. For full details of the fields which are used for these links, see the section on [Database tables](#).



Note: The arrows in the diagram above imply a "has a link to" relationship.

7.2 Database tables

This section lists the tables in the AutoTestSQL database, and the fields available within each table together with their data type. For an explanation of the different data types, see [Data types](#).

[ProductFamily](#)

This table contains details of all [Product Family](#) records.

Field name	Data type	Description
productfamily_name	CHAR(50)	Name of the Product Family.
productfamily_id	INT	Unique ID of the Product Family.

[Product](#)

This table contains details of all [Product](#) records.

Field name	Data type	Description
product_name	CHAR(50)	Name of the Product.
product_script	NVARCHAR(1024)	File name of the Product Scriptlet.
product_id	INT	Unique ID of the Product.
productfamily_id	INT	ID of the Product Family that this Product belongs to.

Operator

This table contains details of all [Operator](#) records.

Field name	Data type	Description
operator_id	INT	Unique ID of the Operator.
operator_name	CHAR(30)	Name of the Operator.
operator_userid	INT	User-entered ID of the Operator.
operator_password	CHAR(20)	Password of the Operator. Note that this is encrypted before storage in the database.
operator_changepassword	TINYINT	Whether the Operator must change their password at the next logon.
operator_allowedoperations	SMALLINT	Used to store details of which operations this Operator is allowed to perform.

TestSet

This table contains details of all [Test Set](#) records.

Field name	Data type	Description
testset_id	INT	Unique ID of this Test Set
testset_name	CHAR(50)	The name of this Test Set
testset_script	NVARCHAR(1024)	File name of the Test Set Scriptlet.
testset_parent	INT	ID of this Test Set's parent Test Set in the test structure. This is set to NULL if it is a top-level Test Set.
testset_index	INT	Index of this Test Set into its parent. This allows Test Sets to be correctly ordered within their parents in the test structure.
product_id	INT	ID of the Product that this is a Test Set for. This is set to NULL if it is not a top-level Test Set.
testset_parentname	NVARCHAR(1024)	Full name of all this Test Set's parents, in order of hierarchy, separated by the backslash character (' \ '). This allows Test Reports to be created quickly since the entire test structure does not need to be traversed for each part of the hierarchy.

TestLimits

This table contains details of [limit](#) records for a Test Element.

Field name	Data type	Description
testlimits_id	INT	Unique ID of this limit record.
testlimits_upper	NVARCHAR(1024)	Upper limit for the Test Element.
testlimits_lower	NVARCHAR(1024)	Lower limit for the Test Element.

TestElement

This table contains details of all [Test Element](#) records.

Field name	Data type	Description
testelement_id	INT	Unique ID of this Test Element.
testelement_description	NVARCHAR(1024)	Description of the Test Element.
testelement_index	INT	Index of this Test Element into its parent Test Set. This allows Test Elements to be correctly ordered within their parents in the test structure.
testlimits_id	INT	ID of the TestLimits record containing the limits for this Test Element.
testelement_resulttype	TINYINT	The type of result of this Test Element: 0 = Numerical result 1 = Pass/Fail 2 = String
testset_id	INT	The ID of this Test Element's parent Test Set.
testelement_sigfigs	SMALLINT	The number of decimal places to use when displaying results created for this Test Element.

ProductTestSession

This table contains details of all [Test Session](#) records.

Field name	Data type	Description
producttestsession_id	INT	Unique ID of this Test Session.
producttestsession_datetime	SMALLDATETIME	The date and time that this Test Session was started.
operator_id	INT	ID of the Operator performing this test.
producttestsession_status	TINYINT	The status of this Test Session (See description of the ResultElement table for details).
product_id	INT	ID of the Product that this is a Test Session for.
producttestsession_serialnum	NVARCHAR(20)	Serial number of the Product Instance that this test is being done for.

ResultSet

This table contains details of all [Result Set](#) records.

Field name	Data type	Description
resultset_id	INT	Unique ID of this Result Set.
resultset_parent	INT	ID of this Result Set's parent Result Set. This is set to NULL if it is a top-level Result Set.
testset_id	INT	ID of the Test Set from which this Result Set was created.
producttestsession_id	INT	ID of the Test Session that this Result Set is part of.
resultset_status	TINYINT	The status of this Result Set (See description of the ResultElement table for details).

ResultElement

This table contains details of all [Result Element](#) records.

Field name	Data type	Description
resultelement_id	INT	Unique ID of this Result Element.
resultelement_value	NVARCHAR(1024)	The Result Element's value.
resultelement_upperlimit	NVARCHAR(1024)	The upper limit used to calculate this Result Element's status.
resultelement_lowerlimit	NVARCHAR(1024)	The lower limit used to calculate this Result Element's status.
resultelement_status	TINYINT	The status of this Result Element: 0 = Failed 1 = Passed 2 = Unknown 3 = Non-critical. See How a result status is calculated for details of how limits are applied to get these.
testelement_id	INT	ID of the Test Element that this Result Element was created from.
resultset_id	INT	ID of this Result Element's parent Result Set.
producttestsession_id	INT	ID of the Test Session that this Result Element is part of.
resultelement_numtests	TINYINT	The number of times that this Result Element has been tested. This will usually be 1, unless you have specified that you wish to re-test failed results (see Re-testing failures), and have used the Options dialogue box to specify that you wish to keep a record of which Result Elements have been re-tested.

FaultDetails

This table contains details of all fault details stored in the database.

Field name	Data type	Description
faultdetails_id	INT	Unique ID of this FaultDetails record.
faultdetails_fault	NVARCHAR(1000)	Description of the fault.
faultdetails_board	NVARCHAR(20)	The board containing the component that caused the fault.
faultdetails_component	NVARCHAR(20)	The component that caused the fault.
faultdetails_solution	NVARCHAR(1000)	Description of how the fault was fixed.
faultdetails_comment	NVARCHAR(1000)	Any comments about this fault
resultelement_id	INT	ID of the Result Element that these Fault Details are linked to.
operator_id	INT	ID of the Operator that entered these fault details. Note that this may be different to the one who carried out the test in the first place, i.e. the Operator linked to the ProductTestSession record.

NextID

As seen above, many of the records in the AutoTestSQL database contain a field holding a unique record ID number (for example product_id in the Product table). This number is used to link records to other tables.

In order to ensure that these ID numbers are kept unique, the NextID table holds a single record that stores the next unused ID number in sequence for each of these record types. Every time a new record of a given type is created by the AutoTestSQL application, it is given the next ID number from the correct column in this table, and the number in this table is incremented.

Field name	Data type	Description
nextid_testset	INT	The next unique ID for a TestSet record.
nextid_testelement	INT	The next unique ID for a TestElement record.
nextid_testlimits	INT	The next unique ID for a TestLimits record.
nextid_producttestsession	INT	The next unique ID for a ProductTestSession record.
nextid_resultset	INT	The next unique ID for a ResultSet record.
nextid_resultelement	INT	The next unique ID for a ResultElement record.
nextid_productfamily	INT	The next unique ID for a ProductFamily record.
nextid_product	INT	The next unique ID for a Product record.
nextid_operator	INT	The next unique ID for a Operator record.
nextid_faultdetails	INT	The next unique ID for a FaultDetails record.

VersionInfo

This table contains the current version information for the database.

Field name	Data type	Description
versioninfo_version	INT	Version number for the database. If the database does not match the version required by the AutoTestSQL software, then AutoTestSQL will not run.

Data types

The following table describes each of the data types used in the above tables:

Data type	Description
INT	A 4-byte integer. It can have values from -2,147,483,648 to 2,147,483,647.
SMALLINT	A 2-byte integer. It can have values from -32,768 to 32,767.
TINYINT	A 1-byte integer. It can have values from 0 to 255.
SMALLDATETIME	Date/time data from January 1st, 1900 to June 6th, 2079, stored to the nearest minute.
CHAR(<i>n</i>)	Fixed-length character data with a maximum length of <i>n</i> characters.
NVARCHAR(<i>n</i>)	Variable-length character data with a maximum length of <i>n</i> characters.

7.3 Using SQL from the command-line

Setting up a shortcut to MSDE

If you want to perform simple SQL operations on the database, you can set up a simple shortcut on your desktop:

- 1) Right-click on the desktop and select "New Shortcut" from the resulting menu.
- 2) Under the field titled "Type the location of the item", enter **osql -U sa -q "Use AutoTest;"**
- 3) Click the [Next >] button.
- 4) Enter a sensible name for the shortcut, such as "MSDE - AutoTestSQL"
- 5) Click Finish to create the shortcut on the Desktop.

Using the shortcut

When you double-click on the shortcut created above, it will open up a command-line window using the AutoTestSQL database, and will prompt you for the password. This is the password of the system administrator account (the account whose login is **sa**).

NB: When MSDE is first installed, this password is blank, but it is advisable to change it as soon as possible. See [Changing the database administrator password](#) for details of how to do this.

As you enter the password, the characters you type will not be displayed on the screen; the display will not change until you press <Enter>. Once the correct password has been entered (the command-line shortcut will close if an incorrect password is used), the prompt will change to show **1>**. This means that it is ready to start accepting SQL commands. You can type in commands that go over more than one line and then execute them using the **GO** command.

Type **exit** to close the command-line window.

7.4 Changing the database administrator password

When [MSDE](#) is first installed, it has a system administrator whose login name is **sa** and whose password is blank.

This system administrator has complete access to MSDE, including the AutoTestSQL database and all other databases. For this reason it is important to change the password to prevent unauthorized access to the database.

Since MSDE is a cut-down version of SQL Server, there are no graphical tools to aid you with this. Changing the password must be done using the command-line. If you have set up command-line access to the AutoTestSQL database (see [Using SQL from the command-line](#)), then you can simply double-click on the icon created, press <Enter> to enter a blank password, and then type the following:

```
sp_password NULL, newpassword, sa
go
```

where **newpassword** is the your new password.

Otherwise, click on the Windows [Start] button, and select the **Run** option. Type the following:

```
osql -U sa "sp_password NULL, newpassword, sa"
```

where **newpassword** is your new password. You will be prompted for the existing password; just press <Enter> and the password will be changed.

Note that the above commands both assume that you are changing the password from **NULL** to **<newpassword>**. To change the password in the future, you will need to replace the **NULL** in both

the commands above to be the password that you have just entered.



All the SQL files and batch files supplied with AutoTestSQL, together with any that may be provided in the future by Prism Sound, use the 'sa' login by default. When run, they will prompt you for the password.

7.5 Creating new database logins

When **MSDE** is first installed, it has only a single system administrator user (**sa**), and a user created specifically for the AutoTestSQL software to access the database. For creation of management reports, however, it is recommended that you create at least one other user for the report designer to use; preferably a user with read-only access.

To create another user, you will firstly need to set up command-line access to the database as the system administrator user. See [Using SQL from the command-line](#) for details of how to do this. Once this is done, you can double-click on the icon created and enter the password of the **sa** user to access the database.

To add a new user to the database, you need to type the following lines into this command window:

```
EXEC sp_addlogin '<name>', '<password>', AutoTest
GO
EXEC sp_adduser '<name>', '<name>', '<db_access>'
GO
```

where <name> is the name of the new user, and <password> is their password. <db_access> is the level of access to the database that you want to give the user, and should be one of the following:

db_owner	Can perform any activity in the database.
db_backupoperator	Can back up the database
db_datareader	Can see any data from all user tables in the database.
db_datawriter	Can add, change or delete data from all user tables in the database.



There are other database roles available, but they are beyond the scope of this manual. Knowledge of these roles is not necessary for use with the AutoTestSQL database.

7.6 Backing up and restoring the database

From the moment that you have started to enter a test structure in the database, it is *extremely* important that you back your data up on a regular basis.

Since **MSDE** is a cut-down version of SQL Server, it does not have a graphical interface to help with this task. However, two batch files are supplied with AutoTestSQL to help you to back up and restore the database. You may wish to edit these files to suit your needs.

These batch files run a copy of the `osql` command-line utility together with the relevant command. This means that they can be run simply by double-clicking on the file.

Backup batch file

The backup batch file, **MSDE_BackupDatabase.bat**, executes one of the following commands:

```
BACKUP DATABASE AutoTest
TO DISK = '.\AutoTest_backup'
WITH INIT
```

or

```
BACKUP DATABASE AutoTest  
TO DISK = '.\AutoTest_backup'
```

The first of these commands overwrites the existing database backup; the second appends the current database to the file.

NB: Both these files overwrite the backup file itself, "AutoTest_backup". If you want to keep subsequent backup files, it is a good idea to perform this backup, then rename the file to something unique before the next backup is performed.

Restore batch file

The restore batch file, **MSDE_RestoreDatabase.bat**, executes the following command:

```
RESTORE DATABASE AutoTest  
FROM disk = '.\AutoTest_backup'  
WITH REPLACE
```

this replaces the current database with the one contained in the "AutoTest_backup" file.



Use this batch file with care! It will NOT give you any warning that your data will be overwritten, and will not ask you if you are sure!

7.7 Multi-user setup

AutoTestSQL can be used in a multi-user environment, although the number of users will be limited. See [Limitations of MSDE](#) for further details.

Once set up for multiple users, a single Server machine can be used to store the database, scripts etc and various AutoTestSQL Clients can then run the software to perform test setup, run tests, etc.

To set up AutoTestSQL for multiple Operators, you will need to take the following steps:

On the Server machine

- 1) Run the SQL Server Network Utility, **svrnetcn.exe**.
This should be located in the SQL Server Tools\Binn folder, which will be installed by default to **C:\Program Files\Microsoft SQL Server\80\Tools\Binn**
Move "TCP/IP" from the list of disabled protocols, to the list of enabled protocols.
- 2) Stop and Restart the SQL Server Service (use the icon in the bottom right of the taskbar).

On each Client machine

- 1) Install the AutoTestSQL software using the **setup.exe** file from the installation CD.
- 2) Open the **ODBC Sources** Administrative tool (Start\Program Files\Administrative tools\Data Sources (ODBC)).
- 3) Select the "AutoTest_MSDE" data source, and click the [Configure] button.
- 4) Under "Which SQL Server do you wish to connect to?", select the name of the computer you're using as the Server. Click [Next].
- 5) Under "How should SQL Server verify the authenticity of the login ID?", select "With SQL Server Authentication".

6) Click on the [Client Configuration] button, and ensure that "TCP/IP" is selected under "Network libraries". Click [OK].

7) UN-check the "Connect to SQL Server to obtain default settings" box.

NB: This ODBC Source is simply a vehicle for the AutoTestSQL software to access the database, and should NOT contain login details of its own.

If you don't uncheck this box, you will have to enter the database administrator's "sa" name and password to continue, thereby allowing anyone full access to the database (with administrator privileges) via this ODBC source!

8) Continue through the rest of the setup dialogue boxes, clicking [Next] at each one.

Note that when running the AutoTestSQL software on each Client machine, you will have to alter the Options to ensure that the various folders for Scripts, Management reports etc point to the correct place. You will probably decide to keep all your Scripts and Reports on the Server, so you'll need to ensure that the folders point to the Server machine rather than the Client machine.

7.8 Limitations of MSDE

The **MSDE** database is a cut-down version of SQL Server. It has certain size and performance limitations, as well as a lack of a graphical interface.

MSDE is a sufficiently functional database to be used initially with AutoTestSQL. However, in a production environment it may be necessary to upgrade to Microsoft SQL Server to allow tests to be performed without size or performance limitations.

Some of the limitations of MSDE are outlined below:

- Limited to 2GB of data.
- Limited to 5 concurrent users.
- No graphical tools for database administration.

If any of the above are likely to be a problem, then an upgrade may be necessary to SQL Server at some point in the future.



It is intended that AutoTestSQL will be improved to allow it to work with the Open Source database, MySQL. However in its current release version, MySQL's functionality is too limited to work with the AutoTestSQL software.

Part



8

Management reporting

8 Management reporting

AutoTestSQL has no built-in way of creating and viewing management reports. It is anticipated that a professionally available reporting tool will be used to create reports, since these have a huge amount of useful functionality to format reports in any way you choose.

Several reporting tools are available that can connect to a [SQL](#) database and extract information in the way that you select. The most common of these is Crystal Reports, available from www.businessobjects.com. A 30-day trial download of Crystal Reports is available from this web site.

We have provided a Report Viewer application with AutoTestSQL, which is optionally selectable as part of the AutoTestSQL installation. This is a simple application that lists any Crystal Report files in a specified [folder](#) and allows you to view them. Provided with this application are a few simple report templates that show you some of the things possible with Crystal Reports. Please note that these are by no means extensive, and are simply provided to show the kind of reports that can be produced. These templates can be altered if you have your own copy of Crystal Reports.

See [The AutoTestSQL Report Viewer](#) for full details of the report viewer.

To find out how to connect to the AutoTestSQL database from your chosen report designer, see [Connecting to the database](#).

8.1 Connecting to the database

Whichever report tool you use to design reports, it will need to connect to the database. The best way of doing this is probably through an [ODBC](#) data source.

A simple MSDE ODBC data source is supplied for use with the AutoTestSQL software called AutoTest_MSDE. However, this data source does not have any details of the user login and password required to access the database, so it cannot be used from outside the AutoTestSQL software. For use with a report design tool, you will have to create a new data source that contains the login details of a database user allowed to access the database for reporting purposes.

It is probably not a good idea to use the database system administrator (**sa**) for this purpose, since this may allow anyone accessing the report to get full access to the database. You should create a new database login to create and view reports - this login only needs read permission on the database, since creating and accessing reports does not require making any changes to the database itself. See [Creating new database logins](#) to find out how to set up a new database user for report access.

[Creating an ODBC connection](#)

Once you have a database login for reports, you should create an ODBC data source. To do this, select **Data Sources (ODBC)** from the Administrative Tools option in the Windows Control Panel. Ensure that the **User DSN** tab is selected, and click the **[Add]** button. Select "SQL Server" from the list of drivers, and click **[Finish]**.

You now need to set up the details of this ODBC Source. The Name of the source can be anything sensible; for example "AutoTestSQL Reporting". The Description can be left blank, and the SQL Server must be selected as the machine containing the AutoTestSQL database. This will probably be "(local)".

Click **[Next >]**, and select "With SQL Server authentication..." at the top of the next dialogue box. At the bottom, ensure that "Connect to SQL Server..." is checked, and enter the name and password of the database user that you wish to use to create and access reports.

After clicking **[Next >]**, ensure that "Change the default database to ..." is checked, and that AutoTest

is selected as the database to change to. You can now select [Next >] again, and finally [Finish] to set up the data source. If you wish, you can click [Test Data Source] on the last dialogue box to check your entries, before clicking [OK] to actually create the data source.

This data source can then be used from within your chosen report designer to access the AutoTestSQL database.

8.2 The AutoTestSQL Report Viewer

The AutoTestSQL Report Viewer is a simple application provided with AutoTestSQL. It allows viewing of example report templates created using Crystal Reports. As well as viewing the examples, it can be used as a simple viewer for any report templates subsequently created using Crystal Reports.

For an overview of the Report Viewer's user interface, see [User interface basics](#).

For a detailed list of the menu options available, see [Menus](#).

For a description of the customizable Options available for the Report Viewer, see the [Options dialogue box](#).

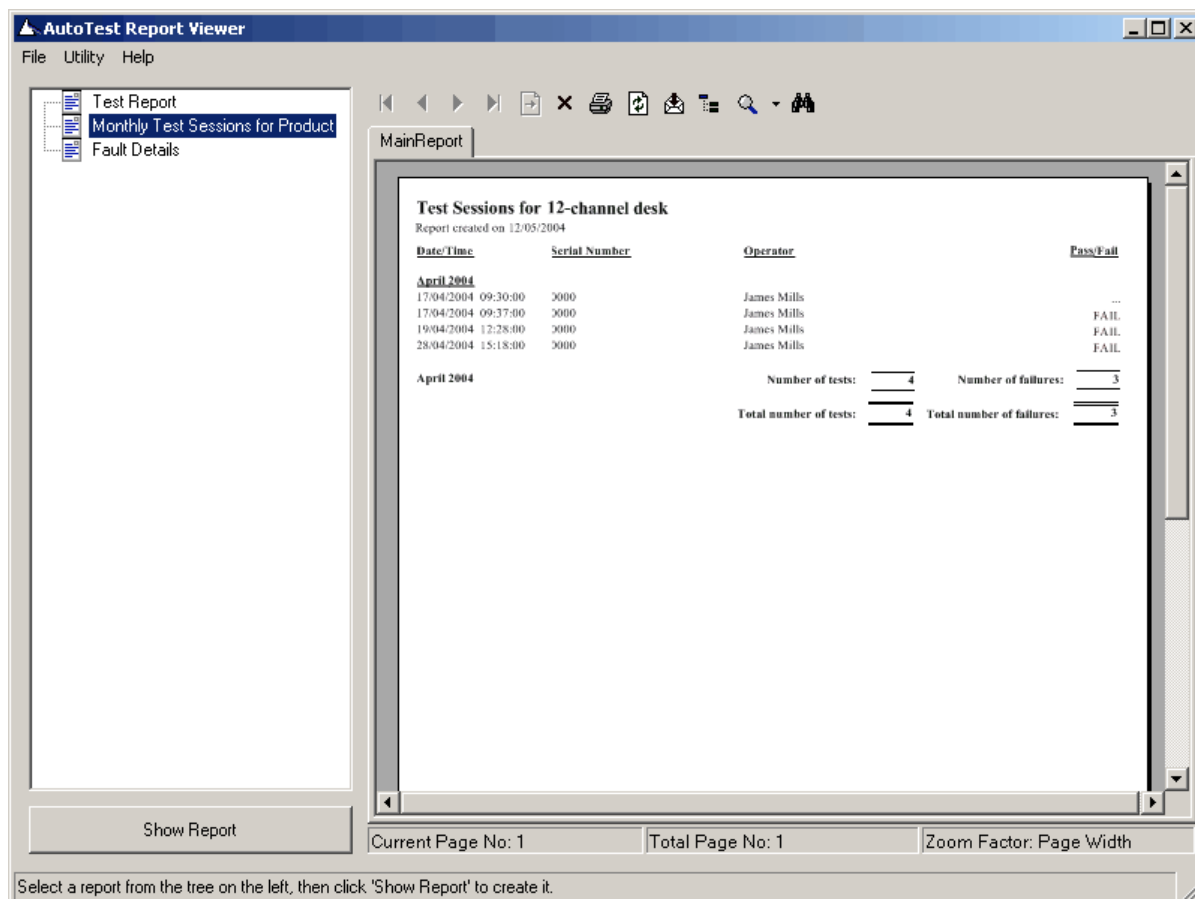
For details of the embedded Crystal Report viewer control, see [The Crystal Report viewer](#).

To find out how to use the Report Viewer to view your own reports, see [Creating your own report templates](#).

8.2.1 User interface basics

The AutoTestSQL Report Viewer's user interface consists of a number of basic elements:

- [Menu bar](#)
- [List of report templates](#)
- [Crystal Report viewer](#)
- [Status bar](#)



Menu bar

The Menu bar is situated at the top of the Report Viewer window. It provides access to functions available within the Report Viewer. See [Menus](#) for full details of the contents of the menus.

List of report templates

The left hand side of the Report Viewer screen contains a list of all the Crystal Report templates contained in a specified folder. The location of this folder can be entered using the [Options dialogue box](#).

The [Show Report] button will load the selected report template from this list and show it in the Crystal Report viewer on the right.

Crystal Report viewer

The right hand side of the Report Viewer screen contains an embedded viewer for Crystal Reports templates. It will view the selected report from the tree on the left. See [The Crystal Report viewer](#) for full details of each of the options available.

Status bar

The bottom line of the Report Viewer window is the Status bar. This shows important indications of the current state of operation, including warning messages.

8.2.2 Menus

The following menu options are available in the AutoTestSQL Report Viewer:

File menu

Exit Exits the Report Viewer application.

Utility menu

Options... Opens the [Options dialogue box](#), from which you can change various operating options for the Report Viewer.

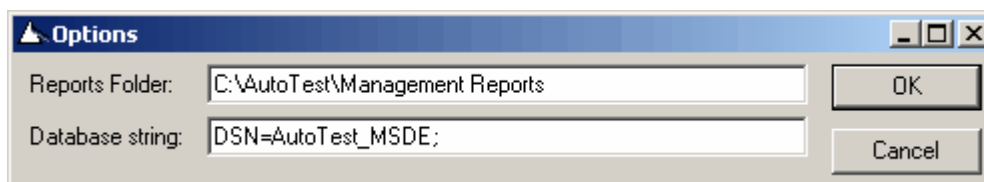
Help menu

Help Contents Accesses the contents page of the on-line help file.

**About
AutoTestSQL
Report Viewer...** Displays a box describing the current AutoTestSQL Report Viewer software release.

8.2.3 Options dialogue box

The Options dialogue box is available from the Report Viewer's Utility menu. It allows the Operator to specify a variety of miscellaneous operating options for the Report Viewer. These are stored in the Windows [registry](#), and are thus retained for all future sessions.



Reports folder: This field defines the location of the folder containing the Crystal Reports report templates. If this is changed, and the [OK] button pressed, then the list of report templates on the main Report Viewer screen will be refilled.

Database string: This field contains the string used to connect to the database. By default this contains a string used to open an [MSDE](#) database on the local machine, i.e:
`DSN=AutoTest_MSDE;`



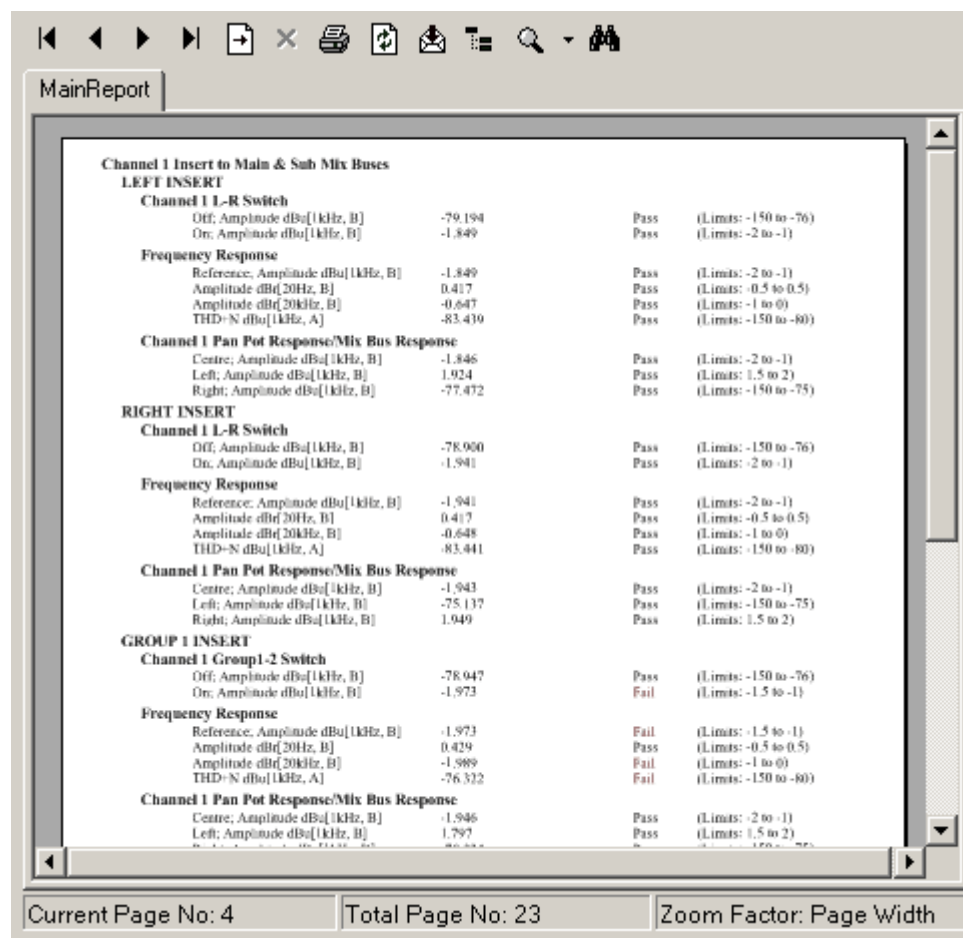
Since this is accessing the same database as the AutoTestSQL software, the Database string is stored in the same place in the registry for both AutoTestSQL and the Report Viewer. Changing this value will also affect the AutoTestSQL software's database opening details.

OK: Applies any changes made and closes the Options dialogue box.

Cancel: Exits the Options dialogue box without making any changes.

8.2.4 The Crystal Report viewer













The Crystal Report Viewer is an embedded control that the Report Viewer uses to display Crystal Report templates.



This control consists of a [Toolbar](#), a tabbed area showing the report, and a Status bar detailing the current page, total number of pages and current zoom factor.

The Crystal Viewer toolbar

The following options are available from the Crystal Report viewer Toolbar:

	First page: Takes you to the first page in the report. This is disabled if the current page is the first or only page.
	Previous page: Takes you to the previous page in the report. This is disabled if the current page is the first or only page.
	Next page: Takes you to the next page in the report. This is disabled if the current page is the last or only page.
	Last page: Takes you to the last page in the report. This is disabled if the current page is the last or only page.
	Goto page: Allows you to select a page number of the report to display.
	Close current view: Closes the currently selected view of the report.
	Print report: Prints the current report.
	Refresh: Redisplays the current report. NB: If the report requires parameters to be specified, the Report Viewer will ask the user for these parameters to be entered again.
	Export report: Exports the current report.
	Toggle Group Tree: Some reports have the ability to select a portion of the report and create a sub-report for it. This toggles a tree of possible sub-reports that can be selected from the main report.
	Zoom: Drops a list of zoom options for you to select.
	Search Text: Allows you to search the report for specific text.

8.2.5 Creating your own report templates

When designing a Crystal Report template, the template can be set up to require parameters to be entered to specify exactly which information should be included in the report. When the report template is viewed, the Operator is prompted for these parameters. These parameters are then applied to the database [query](#) to ensure that the correct subset of information is loaded from the database.

Under normal circumstances, Crystal Reports would automatically bring up its own dialogue box to prompt the user for *each* database parameter in turn. This can be messy and time-consuming, however, so the AutoTestSQL Report Viewer detects which parameters are needed and prompts for the parameters in its own way. It looks for specific names for these parameters, and if it finds them, uses its own prompts to request the information from the user. See [Report parameter names](#), below, for details of the parameter names automatically detected by the Report Viewer.

NB: The example Crystal Report templates that are supplied with the AutoTestSQL Report Viewer all require the Operator to enter parameters to limit the report's information.

Selection of a Test Session

If the report is for a specific Test Session, for example to create a single Test Report, then the Operator must be asked to select enough information to allow a single Test Session to be retrieved from the database. In this case, the following dialogue box will be displayed:

Select Test Session

Product: 12-channel desk

Operator: James Mills

Serial number:

Test session:

- 12-channel desk: Serial no. 0000, tested on 17/04/2004 at 09:30 by James Mills (...)
- 12-channel desk: Serial no. 0000, tested on 17/04/2004 at 09:37 by James Mills (Failed)
- 12-channel desk: Serial no. 0000, tested on 19/04/2004 at 12:28 by James Mills (Failed)**
- 12-channel desk: Serial no. 0000, tested on 28/04/2004 at 15:18 by James Mills (Failed)

OK Cancel

Selection of a Product, Operator, serial number or any combination of the three will result in a list of all the applicable Test Sessions being displayed in the list at the bottom of this dialogue box. The appropriate Test Session can then be selected, and the [OK] button clicked to display the report.

Selection of other parameters

If the report is not for a single Test Session, then it may require one or more parameters to limit the data included in the report. In this case, the Report Viewer will display a dialogue box that asks for each of the parameters in turn. For example, a report which requires a 'Product' parameter, a 'Start date' parameter and an 'End date' parameter would show the following dialogue box:

Enter Report parameters

Product: 12-channel desk

Start Date: 01 April 2004

End Date: 30 April 2004

OK Cancel

Once the required options have been selected, the [OK] button can be clicked to show the report.

Report parameter names

When designing a report template, you can use any of the following parameter names to get the AutoTestSQL Report Viewer to prompt the user in one of the above ways:

<u>Parameter name</u>	<u>Description</u>
ProductTestSessionID	Prompts the user for a single Test Session record. See Selection of a Test Session , above. NB: This parameter cannot be used in conjunction with any of the other parameters in this list.
ProductFamilyName	Prompts the user to select a Product Family from a drop-list, and uses its name as a parameter to the report.
ProductFamilyID	Prompts the user to select a Product Family from a drop-list, and uses its unique record ID as a parameter to the report.
ProductName	Prompts the user to select a Product from a drop-list, and uses its name as a parameter to the report.
ProductID	Prompts the user to select a Product from a drop-list, and uses its unique record ID as a parameter to the report.
OperatorName	Prompts the user to select an Operator from a drop-list, and uses its name as a parameter to the report.
OperatorUserID	Prompts the user to select an Operator from a drop-list, and uses its Operator ID as a parameter to the report.
OperatorID	Prompts the user to select an Operator from a drop-list, and uses its unique record ID as a parameter to the report.
SerialNum	Prompts the user to enter a serial number, and uses it as a parameter to the report.
StartDate	Prompts the user to select a start date from a Windows date/time picker control, and uses it as a parameter to the report.
EndDate	Prompts the user to select an end date from a Windows date/time picker control, and uses it as a parameter to the report.
Date	Prompts the user to select a single date from a Windows date/time picker control, and uses it as a parameter to the report.

Part



9

Glossary

9 Glossary

.

.css file - See [Cascadable Style Sheet](#).

A

ActiveX - ActiveX describes a group of technologies (incorporating [OLE](#), [automation](#) etc) by which different Windows applications can communicate with each other and use each others' capabilities.

ActiveX Control - An ActiveX Control is a control that can be created in a script, and then controlled by accessing properties and methods that it exposes through an [ActiveX](#) interface.

Administrator - An [Operator](#) that has full access to all parts of the AutoTestSQL application.

Automation - Automation is the process by which one application can control another using its [ActiveX](#) interface.

B

Beta - A preliminary or testing stage of a software or hardware product.

Boolean - A Boolean variable in a script is one that can contain the values **True** or **False**.

C

Cascadable Style Sheet - A file used in conjunction with [HTML](#) files. It defines various styles that can then be referred to from the HTML file, enabling the style of fonts, etc to be changed without changing the HTML file itself.

Configuration - A dScope Configuration is a file containing the settings of the dScope. The state of the dScope settings can be saved as a Configuration, and then re-loaded as a known starting point for a test.

Constant - A value defined in a script that cannot change.

Control Panel - A program used to change settings in the Windows operating system. It allows changing of items such as keyboard and mouse settings, display colours and resolution, and modem, network and printer settings.

D

Database - A collection of information that is organized so that it can be easily accessed, managed and updated. The AutoTestSQL database is a [relational database](#).

Desktop - The background area of your computer screen, where icons for some applications appear.

Double-precision - A floating-point number occupying 8 bytes (64 bits). It can have values from -1.79769313486232e+308 to 1.7976931348623158e+308.

dS-NET - A proprietary serial interface protocol used to connect peripherals such as I/O Switchers to the dScope.

E

EUT (Equipment Under Test) – The device being tested by the dScope in conjunction with AutoTestSQL.

Event - A causal occurrence for a script. An event is said to be *fired* when something occurs to trigger it, and a corresponding [subroutine](#) or [function](#) in the script is run.

F

Failed result - A result written to the AutoTestSQL database that, depending on its type, has failed. For example a numerical result is failed if it is outside the Test Element's specified limits; a pass/fail result is failed if it is Fail; a string result is failed if either limit is specified and does not match the result string.

Folder - A named storage area on the computer containing files and other folders. For example, in "C:\Program Files\Prism Sound\AutoTest", "Program Files" is a folder.

Function - A named group of statements within a script, that can be called from elsewhere in the script to perform a certain action or group of actions. Functions are similar to [subroutines](#), but return a value to the part of the script from which they were called.

G

H

Hex (hexadecimal) - A convention for conveniently representing binary data such as digital audio samples. Each four-bit 'nibble' of the binary word is represented by a character indicating its value: 0..9,A..F. For example, the 24-bit binary value 000000010010110111101111 would be represented in hex as 012DEF. Hex is available as an amplitude unit throughout dScope in order to facilitate some digital measurements.

HTML (HyperText Markup Language) - This is a language used for creating documents, particularly web pages. It consists of a set of symbols or codes inserted in a file that tells web browsers how to display the page's words and images to the user.

Hungarian notation - This is a system used by some programmers, in which the type of a variable is specified by inserting one or more letters at the beginning of the variable name. For example, iReturnValue means that the variable indicating the return value is of type [integer](#), shown by the i at the start

I

I/O Switcher - The I/O Switcher is a [dS-NET](#) device, primarily intended for use with the dScope Series III test and measurement system. It is a serially-controlled 16-into-2 relay switcher, which can be used for switching either analogue or digital (AES3) audio signals. See the documentation accompanying the dScope software for full details of the I/O switcher.

Integer - A 4 byte whole number. It can have values from -2,147,483,648 to 2,147,483,647.

J

K

L

Limit - A Limit can be entered for a Test Element in AutoTestSQL. When the corresponding result is read by the script, this limit is applied and used to calculate the Pass or Fail status of the result. See [How a result status is calculated](#) for details.

M

Mode - A state of operation of the AutoTestSQL software. Setup mode allows entry and editing of Products and tests; Test mode allows these Products to be tested; and Fault-find mode allows test failures to be analyzed and the reasons for failure investigated.

MSDE (Microsoft SQL Server Desktop Engine) - MSDE is a royalty-free, redistributable database engine that is fully compatible with Microsoft SQL Server. It has certain limitations - see [Limitations of MSDE](#) for details.

Multi-tone testing - A method of testing where a large number of discrete tones are used to stimulate the [EUT](#) simultaneously. By capturing a single data set of the output of the EUT, many measurements can be calculated simultaneously. These can include scalar results such as noise, distortion etc. as well as graphical plots against frequency, such as frequency response, distortion spectrum etc. This method is much faster than traditional methods, which would require stimuli to be changed for each scalar measurement and stepped through many frequencies for each plot. The dScope has a uniquely user-friendly way of setting up multi-tone tests.

[N](#)

Non-critical result - A result written to the AutoTestSQL database that has no limits associated with it. Since no limits exist to compare with the result value, it is treated as non-critical and does not affect the overall status of a test.

Notepad - A program included in the Windows operating system that lets you view and edit text files.

[O](#)

ODBC (Open DataBase Connectivity) - A standard developed by Microsoft for accessing different databases from Windows.

OLE (Object Linking and Embedding) - This is the mechanism by which dScope can be remotely controlled via [VBScript](#).

Open Source - Software where the source code is available for anyone to extend or modify. Linux is the best-known example; MySQL is an example of an Open Source [database](#).

Operator - A user of the AutoTestSQL application.

[P](#)

Passed result - A result written to the AutoTestSQL database that, depending on its type, has passed. For example a numerical result is passed if it is within the Test Element's specified limits; a pass/fail result is passed if it is Pass; a string result is passed if either limit is specified, and matches the result string.

Persistent - A Scriptlet is said to be persistent if it remains in memory throughout a test. For example, the Product Scriptlet is persistent, whereas individual Test Set Scriptlets are not, as they are loaded and run when necessary.

Pixel - A single dot on the screen. If the screen resolution is set up to be 800 x 600, this means that there are 800 pixels across the screen, and 600 down.

Product - An item of equipment that can be tested using AutoTestSQL.

Product Family - A group of Products.

Product Instance - An individual device being tested, or an instance of a [Product](#) entered into AutoTestSQL.

Program folder - The [folder](#) containing the AutoTestSQL application (**AutoTestSQL.exe**).

[Q](#)

Query - A query is a request for information from a database. It usually takes the form of a SELECT statement.

R

Reading - A Reading can be created from a numerical measurement in the dScope software by dragging it off its home dialogue box. This allows a flexible representation of the measurement - for example, Readings can be resized, user-coloured, and can have [Limits](#) and bar graphs attached to them.

Registry - A database of information used by Windows to store program information, associations, hardware information, etc.

Relational database - A collection of data organized as a set of formally-described tables from which data can be accessed in different ways without having to reorganize the tables.

Result Set - A grouping of results (Result Elements, or other Result Sets) in the AutoTestSQL database.

Result Element - A single numerical, [boolean](#) or string result in the AutoTestSQL database.

S

Scope - The scope of a variable in a script indicates from where in the script it can be accessed. If a variable is defined within a [function](#) or [subroutine](#), then its scope is within that function only, i.e. it cannot be used outside the function. If it is defined outside a function, it is a global variable, and its scope is the entire script.

Script - See [VBScript](#).

ScriptDlg ActiveX Control - An [ActiveX Control](#) provided with the dScope software to improve the user interface options of the VBScript programming language.

Scriptlet - A file of [VBScript](#) code, attached to a [Product](#) or [Test Set](#), containing a subset of the tests required for a Product.

SQL (Structured Query Language) - A language used by relational databases to query, update and manage data.

Status bar - An area at the bottom of the AutoTestSQL, dScope or Report Viewer screen which displays various information about the current state of the system.

Subfolder - A [folder](#) that is underneath another folder in the folder structure on a computer. For example, in "C:\Program Files\Prism Sound\AutoTest", "Prism Sound" is a subfolder of "Program Files", and "AutoTest" is a subfolder of "Prism Sound".

Subroutine - A named group of statements within a script, that can be called from elsewhere in the script to perform a certain action or group of actions. Subroutines do not return a value.

Sweep - A sequence of individual measurements made whilst varying a parameter of the stimulus. For example a frequency response sweep would be made by measuring the gain of an EUT whilst varying the frequency of the stimulus. dScope provides a versatile sweeping capability, wherein many different Generator parameters can be varied whilst plotting up to four simultaneous results. Many tests which have traditionally been frequency-swept are now better performed using [multi-tone](#) techniques, which are much faster.

T

Table - The basic [database](#) storage object consisting of a collection of rows (records) divided into columns (fields) that contain data.

Test Script - The script that is created automatically by the AutoTestSQL software from a Product Scriptlet and all the Product's Test Set Scriptlets. This script is then run to perform the tests on the Product.

Test Set - A group of [Test Elements](#) or other Test Sets used to define the hierarchical structure for a [Product](#)'s tests in AutoTestSQL.

Test Element - An individual part of a test that corresponds to a single result value.

Test Session - A Test Session for a Product is a test done on a particular [Product Instance](#) at a specific date and time. It contains one or more [Result Sets](#), which in turn may contain other Result Sets and [Result Elements](#).

Thread - A Windows program can contain one or more threads. This allows several tasks to be completed at the same time - In AutoTestSQL, for example, one thread can be writing results to the database, while another is running a new test.

Title bar - A window's title bar is the bar at the top of the window, containing the window name, and usually buttons to minimize, maximize and close the window.

Toolbar - A bar in the AutoTestSQL, dScope or Report Viewer user interface containing a variety of 'icons' which can be clicked as shortcuts to various functions.

[U](#)

Unknown result - A result written to the AutoTestSQL database, whose pass or failure state cannot be determined by applying [limits](#). This can occur when Result Elements are first created in the database but before the result value has been measured.

User Script - A script whose contents are defined by the user. It is automatically referenced by the AutoTestSQL software and can contain [constants](#), [variables](#), [functions](#) and [subroutines](#) which are then made available to all Scriptlets for any Products or Test Sets.

[V](#)

Variable - A named storage location in a [script](#) containing data that can be modified during program execution. Each variable has a name that uniquely identifies it within its level of [scope](#).

VBScript - A script or program in the Microsoft Visual Basic Scripting language which performs a test or series of tests.

[W](#)

[X](#)

[Y](#)

[Z](#)

Index

- . -

.css file 109

- A -

ActiveX 109
ActiveX Control 23, 109
Administrator 109
 AutoTestSQL Operator 82
 Database 93
Alignment 42
Allowed operations 61
Automation 109
 dScope Series III 10
AutoTestSQL database 87
AutoTestSQL Report Viewer 100

- B -

Backing up the database 94
Beta 109
Boolean 109

- C -

Cascadable Style Sheet 70, 109
Change Operator 51, 63, 72
Change password 82
 At next login 61
 Database administrator 93
Code style 44
ColourMsgBox 37
Colours 42
Command-line 93
Comments 44
Common script problems 46
Completion of test 68
Configuration 109
Constant 109
 Qualifying 46
 Writing scripts 44
Control Panel 109

Copy 51
 Edit menu 51
 Toolbar 52
Copying 54, 56, 60
 Operator 60
 Test Element 56
 Test Set 56
Creating report templates 104
Crimson Editor 43
Crystal Reports 99
 Report parameters 104
 Report templates 104
 Viewer 103
Cut 51
 Edit menu 51
 Toolbar 52

- D -

Database 87, 109
 Backing up 94
 Connecting to 99
 Restoring 94
Database administrator 93
Database login 94
 Creating new 94
Database string 79
 AutoTestSQL Report Viewer 102
Database structure 87
 Tables 88
Debugging scripts 43, 45
Deleting 54, 56, 60
 Operator 60
 Product 54
 Product Family 54
 Test Element 56
 Test Set 56
Description (Test Element) 59
Desktop 109
Discard test results 79
Discarding results 68
Double-precision 109
dScope Series III 10, 11
 AutoTestSQL and 10, 11
 Script editor 43
dS-NET 109

- E -

Edit menu 51

End Now (stopping a test) 67

Error 45
 in script 45
 Run-time 45
 Syntax 45

Error messages 45

EUT 109

Event 109

- F -

Failed result 109
 Calculation of 18
 Fault-find mode 74
 Selecting 74

FailMsgBox 39

Fault details 76

Fault-find mode 71
 Completion 68
 Considering 26
 Entering fault details 76
 How it works 71
 Overview 10
 Reference 71
 Running selected functions 72
 Stop 72

Fields and controls 63, 72
 Fault-find mode 72
 Test mode 63

File menu 51
 AutoTestSQL Report Viewer 102

File name format 79

Folder 109

Function 24, 109

Function 47

- G -

General tab 79

GetLimits 32

GetMode 33

GetOperatorID 35

GetOperatorName 35

GetProduct 34

GetProductFamily 34

GetSerialNumber 34

Getting started 7

Glossary 109

- H -

Hanging scripts 46

Help 3

Help menu 51
 AutoTestSQL Report Viewer 102

Hex 109

How it works 7

HTML 109
 Test Report 70

Hungarian notation 109

- I -

I/O Switcher 109

ID 35
 Get Operator 35
 Operator 61
 Unique record 88

Integer 109

Introduction 3

- L -

Licensing 11

Limit 109
 Calculating result status 18
 Entering 16
 Lower 16, 32
 Retrieving from database 32
 Upper 16, 32
 Entering 59

Line numbers 45

Login dialogue box 82

Loop 46
 Script stuck in 46, 67

- M -

Management reporting 99
 Overview 10

Manual 3

Menu bar 9
 Report Viewer 100

Menus 51
 AutoTestSQL Report Viewer 102

Message box 37
 Coloured 37
 Failures 39

- Mode 109
 - Buttons on Toolbar 52
 - Retrieving from a script 33
 - Overview 10
- Moving 54, 56, 60
 - Operator 60
 - Test Element 56
 - Test Set 56
- MSDE 96, 99, 109
 - Limitations 96
- Multi-tone testing 109
- Multi-user setup 95
- MySQL 96

- N -

- Name 61
 - Operator 61
 - Product 55
 - Product Family 55
 - Test Set 58
- Naming 79
- Naming tab 79
- Non-critical result 109
 - Calculation of 18
- Notepad 109

- O -

- ODBC 99, 109
- OLE 109
- On-line help 3, 51
- OnTestFinish 29
- OnTestStart 28
- Open Source 109
- Operation overview 7
- Operation reference 51
- Operator 61, 109
 - Administrator 82
 - Allowed operations 61
 - Change term 79
 - Changing current 51, 63, 72
 - Copying 60
 - Deleting 60
 - Entering details 61
 - Moving 60
 - Retrieving from a script 35
 - Setting up 60
- Option Explicit 44, 79
- Options 79

- Options dialogue box 79
 - AutoTestSQL Report Viewer 102
- osql 93

- P -

- Parameters 104
 - Crystal Reports 104
 - Report Viewer 104
- Passed result 109
 - Calculation of 18
- Password 82
 - Changing 82
 - Logging in 82
 - Operator 61
- Paste 51
 - Edit menu 51
 - Toolbar 52
- Persistent 109
- Pixel 109
- Print 51, 52
- Print preview 51, 52
- Print Test Report 79
- Product 34, 55, 109
 - Change term 79
 - Deleting 54
 - Entering details 55
 - Entry of in Fault-find mode 72
 - Entry of in Test mode 63
 - Retrieving from a script 34
 - Setting up 54
- Product Family 34, 55, 109
 - Change term 79
 - Deleting 54
 - Entering details 55
 - Entry of in Fault-find mode 72
 - Entry of in Test mode 63
 - Retrieving from a script 34
 - Setting up 54
- Product Instance 109
 - Change term 79
 - Entering details 66, 67
 - Select new 72
- Product scriptlet 19, 24
- Program folder 109

- Q -

- Qualifying constants 46
- Query 109

- R -

- Readability 44
- Reading 109
- Registry 109
- Relational database 109
- Report templates 104
- Report Viewer 100, 103
- Reporting 10
 - Overview 10
- Reports folder 79
 - AutoTestSQL Report Viewer 102
- Reports tab 79
- Restoring the database 94
- Result 74
 - Selecting in Fault-find mode 74
 - Writing to database 25, 31
- Result Element 17, 109
 - Change term 79
 - Selecting in Fault-find mode 74
- Result Set 17, 109
 - Change term 79
- Result status 16, 18, 59
 - Database 88
 - Test Report 70
- Result type 16, 18, 59
 - Database 88
- Results 68
 - Discarding 68
 - Tree of 65
- Re-testing 67
- Reusability 44, 47
- Run selected functions 72
- Run test 63

- S -

- Save Test Report 79
- Save test results 79
- Scope 109
- Screen layout 9
 - Report Viewer 100
- Script 109
- Script Editor 43
- Script functions 28
 - Built-in 31
 - OnTestFinish 29
 - OnTestStart 28

- Test Script 28
- User Script 36
- ValidateSerialNumber 30
- Script problems 46
- ScriptDlg 109
- Scriptlet 24, 109
 - Change term 79
 - Folder 79
 - Product 19, 24, 55
 - Responsibility of 19
 - Test Set 19, 24, 58, 59
- Scripts 23
 - Debugging 43, 45
 - Hanging 46
 - Reusability 47
 - Writing 43, 44
- Security 82
- Select new Product Instance 72
- Selecting a failed result 74
- Serial number 66, 67
 - Entering for a test 66, 67
 - Retrieving from a script 34
 - Uniqueness of 79
- SetResult 25, 31
 - Description 25
 - Reference 31
- Setting up Products 54
- Setting up tests 56, 60
- Setup mode 53
 - Overview 10
 - Reference 53
- Shortcut 93
- Show levels on list of results 63, 72
- Show Test Report 79
- Software release 11
- SQL 93, 109
- SQL Server 96
- Status bar 9, 109
 - Report Viewer 100
 - View 51
- Stop fault-finding 72
- Stop test 63
- Stop test on failure 63, 72
- StopTest 36
- Subfolder 109
- Subroutine 24, 47, 109
- Sweep 109
- System administrator (database) 93

- T -

- Table 88, 109
 - Database 88
- Terminology 7
- Test 36
 - Failed 36
 - Failure 67
 - Repeating 67
 - Run 63
 - Running 66
 - Stop 63
 - Stopped 35
 - Stopping 36, 67
- Test completion 68
- Test Element 16, 59, 109
 - Change term 79
 - Copying 56
 - Deleting 56
 - Entering details 59
 - Moving 56
 - Setting up 56
- Test mode 62
 - Overview 10
 - Reference 62
- Test Report 68, 70
 - Overview 10
 - Printing 79
 - Saving 79
 - Showing 79
- Test Script 23, 109
 - Debugging 45
- Test Session 109
- Test Set 16, 58, 109
 - Change term 79
 - Copying 56
 - Deleting 56
 - Entering details 58
 - Moving 56
 - Setting up 56
- Test Set scriptlet 19, 24
- Test stopped 68
- Test structure 15
 - Knowledge of in Fault-find mode 77
 - Overview 15
 - Scriptlets in 19
- TestFailed 36
- Tests 62
 - Performing 62

- Setting up 56
- TestStopped 35
- Text alignment 42
- Thread 109
- Title bar 109
- Toolbar 9, 52, 109
 - View 51
- Tools menu 51
- Tree of results 65

- U -

- Undefined variable 44, 79
- Undo 51
- Unknown result 109
 - Calculation of 18
- User interface basics 9
 - Report Viewer 100
- User Script 24, 36, 109
- User.vbs 24
- Utility Menu 102
 - AutoTestSQL Report Viewer 102

- V -

- ValidateSerialNumber 30
- Variable 109
 - Undefined 44, 79
- Variable names 44
- VBScript 109
 - Documentation 23, 45
 - Introduction to 23

- W -

- Writing scripts 43, 44