Complete Time Tracking Database Script v3.12 User Manual

Table of Contents

Part I

1 Introduction

1.1 Welcome

Welcome to Complete Time Tracking Database Script

Complete Time Tracking is an advanced time tracking application that enables multiple users to track time to a central database using automatic time recording and manual time entry and allocate their time to user-defined categories, such as customers, projects and tasks.

Database Script is a command line tool that can be used to extract data from or make changes to the Complete Time Tracking database, on a scheduled or ad-hoc basis without the need to install and configure database connection drivers such as ODBC, JDBC or ADO.NET. A graphical configuration editor is provided to assist with creating the configuration script or command line options required by the script tool.

System Requirements
See the System Requirements section for details on the minimum computer requirements to use Complete Time Tracking Database Script.

See also:
System Requirements
Getting Started

1.2 Getting Help

You can access the help documentation for Database Script by pressing the F1 key or selecting Contents from the Help menu in the configuration editor or by selecting the Help item from the program group on the Windows Start menu.

Topic links within the help documentation allow you to quickly jump to a related topic. To use the help documentation you do not need to be connected to the Internet, however in some places you are given the opportunity to browse to a web page via web links. To use them you will need to be connected to the Internet.

The documentation is organized into related chapters. You can also search for particular help topics using the index, or use the search function to search for topics by keyword.

PDF Manual
The help documentation distributed with the Complete Time Tracking Database Script tool is also packaged as a printable user manual in Adobe PDF format, complete with contents, hyperlinks, page thumbnails, and a comprehensive index.

To view the User Manual select User Manual from the Help menu in the Database Script editor or select User Manual from the program group on the Windows Start menu.

Part II

2 About Complete Time Tracking

2.1 Overview

What is Complete Time Tracking Database Script?

Database Script allows you to configure and run scripts against the Complete Time Tracking database without the need to install and configure database connection drivers such as ODBC, JDBC or ADO.NET.

A script is a sequence of commands that perform various actions, such as retrieving or updating data. Scripts can be used to extract custom data that is otherwise not possible through the pre-defined reports in the Complete Time Tracking user interface. This data can be used in other reporting tools or applications such as Microsoft Excel. Scripts could also be used to add, modify or delete users, categories, time entries and other data programmatically.

Scripts support several commands such as SQL statements, output statements, conditional execution, and the use of variables that can be saved between runs allowing the following run to resume where the last run finished.

Feedback

We welcome any feedback that you may have about Complete Time Tracking and the Database Script tool. Many of the features and functionality have been designed based on customer feedback.

HOW-TO
Providing Feedback
1. Select Send Feedback from the Help menu in Database Script Editor.
2. Fill out our quick online feedback form.

2.2 What's New

22-Dec-2011 v3.11

New Features
- DML can now be used in SQL Statement commands.
- Date and time tokens can be used in the script and variables filenames.
- Single line comments are supported.

Enhancements
- Added the ability to specify the database role for the user.
- Added the option for literal substitution of variables in SQL Statement commands.
- Statements no longer need to be enclosed in /*BEGIN*/ and /*END*/ markers. The query name has been moved from the /*BEGIN:<name>*/ marker to a new /*NAME:<name>*/ marker.
- Support for multi-line comments improved.
- Additional configuration and command line errors are displayed.
- The date and time token format for filenames has been changed from %xx to #xx for improved compatibility with command line options.

Fixes
- Console based output for the display commands, pause at end of script and non-file data output has been fixed.
- Consistent line endings in output.

06-May-2011 v3.1

- Initial release for Complete Time Tracking version 3.

See also:

Support

Part III

3 Installation, Clients and Servers

3.1 System Requirements

The following are the minimum system requirements to use Complete Time Tracking Database Script.

Windows 2000, XP, Server 2003 Vista, Server 2008, Windows 7	Ubuntu Linux (Edgy 6 or Jaunty 9.04) Wine 0.9.49 or later

Note: Other versions of Ubuntu and other Linux distributions may also work. See Linux Support.

- Intel Pentium III 500 or AMD Duron 500 CPU.
- 128 MB RAM (256 MB recommended).
- 20 MB disk space for the installed program.
- A network connection is required if accessing a database located on another computer.
- In addition to the above requirements we recommend that you use a screen resolution of 800x600 or higher.

See also:
Installation

3.2 Installation

Multiple Installations
Multiple installations, different major versions, of Database Script are allowed on a single computer. This allows you to continue to use an earlier version whilst evaluating a newer version.

Complete Time Tracking Server Software
Database Script is used with an existing Complete Time Tracking database. The database is installed with the Complete Time Tracking server software. To access the database you will need to know the server details (as used in the Complete Time Tracking login window) as well as the name and password for a database user. Database users are managed using the Server Manager tool installed with the Complete Time Tracking server software.

The Server computer must be turned on in order for Database Script to be able to access the database and therefore we recommend that you install the Complete Time Tracking server software on a computer that is always powered on. The database server component runs as a Windows Service Application and therefore it functions without a user logged on to the server computer.

Download
You can download the installation programs from the Complete Time Tracking web site download page.

Firewalls
In most cases Database Script requires network access in order to connect to the database. If you have firewall software installed you need to **allow network access** to the Database Script program DatabaseScript.exe.

See also:
System Requirements
Firewall Configuration

3.3 Linux Support

Linux Support

Database Script is supported on Linux. To install on Linux the Database Script software must be installed under Wine (http://www.winehq.org/), a compatibility layer for running Windows programs under Linux. The Complete Time Tracking server is supported on Windows only.

Supported Configurations

The following configuration has been tested:

Ubuntu Edgy 6.10
Ubuntu Jaunty 9.04
Wine 0.9.49

Database Script may work on other Linux distributions. We recommend using the latest stable release of Wine.

Installing Database Script

Installing and Configuring Wine
To install Database Script you must first install Wine. Some Linux distributions have Wine pre-installed. We recommend using the latest stable version of Wine. For instructions on downloading and installing Wine on your Linux distribution see the following web page:

http://www.winehq.org/site/download

Installing Database Script Under Wine
Type "wine SetupProgram.exe" (where SetupProgram.exe is the installation file for Database Script that you downloaded) at the shell prompt/terminal window.

```
$ wine SetupProgram.exe
```

The installation wizard will appear. Follow the steps in the wizard. You should typically install under the emulated C:\Program Files directory.

Wine and Windows Configuration

Windows Version:

Note: Wine should be configured to mimic Windows 2000 for Database Script to work. Windows 98 emulation is known to cause program failures. Run "winecfg" from the shell prompt/terminal window, then on the Applications tab change the Windows Version for "Default Settings" to Windows 2000, or alternatively select Add application and navigate to the installation directory for Database Script in the drive_c\Program Files directory, select DatabaseScript.exe and click Open, then set Windows Version to Windows 2000. Repeat this for DatabaseScriptEditor.exe.

```
$ winecfg
```

Regional Settings:

The date, time and currency formats used in Database Script come from the Windows regional settings. In Windows you typically change this from the Regional and Language Options in the Control Panel, then select Customize, Time, Time format and so on. Under Wine you may be able to change this using one of the following techniques:

a) Run the regional settings control panel applet under Wine from the shell prompt/terminal window:

```
    $ wine control.exe intl.cpl
```

On the Regional Options tab click Customize.

b) Use the Windows registry editor under Wine from the shell prompt/terminal window:

```
    $ wine regedit.exe
```

Navigate to the key HKEY_CURRENT_USER\Control Panel\International key and create or change the following string values:

- sTimeFormat, example value: HH:mm:ss
- sShortDate, example value: MM/dd/yyyy
- sCurrency, example value: $
- sDate, date separator, example value: /
- sTime, time separator, example value: :
- sDecimal, decimal separator, example value: .
- sThousand, thousand separator, example value: ,
- sMonDecimalSep, monetary decimal separator, example value: .
- sMonThousandSep, monetary thousands separator, example value: ,

More Information:

Further information about using and configuring Wine can be found at the following web sites:

http://wiki.winehq.org/FAQ
http://wine-wiki.org/

Uninstalling Database Script

To uninstall Database Script either select Uninstall or Repair from the Database Script group in the Wine Programs menu, or type "uninstaller" from the shell prompt / terminal window, select Database Script and select Remove or Uninstall.

```
    $ uninstaller
```

See also:
System Requirements
Installation

3.4 Firewall Configuration

Running Database Script
The Database Script software communicates with the Complete Time Tracking database on the Complete Time Tracking server computer using a TCP/IP network connection to the default TCP port 18400. The TCP port is configurable using the Complete Time Tracking server manager tool on the Complete Time Tracking server computer and can be specified in the server details if a non-default port is used.

If you are running a corporate firewall computer between the computer on which you install the Database Script tool and the Server computer, or use firewall software on either computer such as Norton Internet Security, ZoneAlarm, McAfee Personal Firewall or Windows firewall, then you must configure the firewall as follows:

- On the Server computer allow the application cttproserver.exe, located in the server\bin sub-folder where the Complete Time Tracking Professional server software was installed, to run as a server listening on TCP port 18400.
- On the computer where Database Script is installed allow the application DatabaseScript.exe,

located in the Database Script installation folder, to connect to the Server computer using TCP/IP.

See also:
Installation

Part IV

4 Getting Started

4.1 Configuration

The Database Script tool, DatabaseScript.exe, is a non-interactive command line program that runs a script containing one or more commands in a script language using the options passed on the command line or stored in a configuration file. The actual script commands can be stored within the configuration file or in a separate text file. The Database Script tool can be run manually from the command prompt, or automatically from a batch file, called from another program, or run as a scheduled task.

Configuration

Several configurable settings control what database and user to use, what script is run, and various formatting options in the output file. These can be specified on the command line or saved in a configuration file. An interactive editor is provided to simplify the process of creating the command line options or configuration file to be used. Alternatively these can be constructed manually or programmatically. When a configuration file is used a single command line option is required to specify the filename and all other command line options are not required.

Mandatory Settings

Setting	Command Line Option	Configuration File Option	Description
Database	-d "database"	<Database>	Identifies the Complete Time Tracking database to connect to. This is in the same format as the server field entered in the Complete Time Tracking login window. See below.
Database user	-u "user"	<DatabaseUser>	The name of the database user (not the Complete Time Tracking user) to connect as. The user determines the access privileges. Users are configured using the Complete Time Tracking Server Manager tool.
Database password	-p "password"	< DatabasePassword >	The password for the database user.
Database role	-r "role"	<DatabaseRole>	The role for the database user. Either GENROLE for limited read only users, ADVROLE for full read only users, or SYSROLE for read-write users.
Output format	-f format	<OutputFormat>	Selects the format or type of data saved to the output file. Options include **CSV** (comma separated fields), **TAB** (tab separated fields), or **XML**. See SQL Statement.

Additionally either the -c or -s command line option is required. See Optional Settings in the following section.

Database Option

The database is specified in the following format:
ServerComputer/TCPPort:DatabaseName

The three components to Database are:
- **ServerComputer**: Mandatory. The host name or IP address of the Complete Time Tracking

server computer.

- **/TCPPort**: Optional. The TCP port of the Complete Time Tracking database server. Only required where the default port 18400 has been changed using the Server Manager tool.
- **:DatabaseName**: Optional. The name of the database. Only required if the default database name has been changed using the Server Manager tool or where multiple databases are used on the server.

Example Database values:

localhost (when the Database Script tool is run on the server computer)
server01
10.0.120.5
server01.company.com
server01/18400
server01:MyDepartmentDatabase
server01.company.com/18400:MyDepartmentDatabase

Optional Settings

Setting	**Command Line Option**	**Configuration File Option**	**Description**
Configuration filename	-c "filename"	n/a	When a configuration file is used this specifies the filename. All settings are stored in the configuration file and no other *command line* options are required.
Configuration file password	-x "password"	n/a	When running a script using a configuration file with the -c command line option and the configuration file is encrypted (encrypted files usually have the dbscx extension) you must provide the configuration file password using this option. The password is case-sensitive.
Script filename	-s "filename"	<ScriptFileName>	When an external script file is used this identifies the file. Include the path if necessary.
Output filename	-o "filename"	<OutputFileName>	When SELECT SQL commands or OUTPUTLINE commands are used their output is saved to this file. If an output filename is not provided the output will be sent to the console. See SQL Statement, OUTPUTLINE.
Variable filename	-v "filename"	<VariableFileName>	When persistent variables are saved for use in a future run they are stored in this file. See LOADVAR, SAVEVAR and DELETEVAR. If omitted a filename is automatically determined.
Append to output file	-a	<AppendToOutputFile>	Defaults to false. If true any output from the script will be appended to the output file. If false the output file will be overwritten if it already exists.
Include query name	**-n** ["prefix"]	<IncludeQueryName>	Defaults to false. Each SELECT SQL command can be given a name which is optionally written to the output file immediately before the data from the SELECT command. This name can be used for example to determine what the following output represents, particularly in the case where there are multiple SELECT SQL

			commands in one database script. The query name option determines if the query name will be written to the output file. See SQL Statement. See Query name prefix for use of the "prefix" option.
Query name prefix	-n **"prefix"**	<QueryNamePrefix>	A text prefix added to the beginning of the query name when written to the output file. This can make it easier to identify the query name programmatically when processing the output file from another program. See SQL Statement. Append the prefix directly to the -n option.
Include header	**-h** ["prefix"]	<IncludeHeader>	Defaults to false. Each SELECT SQL command returns one or more columns (also known as fields) of data. When enabled the include header option writes the names of these columns to the output file immediately before the data. See SQL Statement. See Header prefix for use of the "prefix" option.
Header prefix	-h **"prefix"**	<HeaderPrefix>	A text prefix added to the beginning of the header when written to the output file. This can make it easier to identify the header programmatically when processing the output file from another program. See SQL Statement. Append the prefix directly to the -h option.
Pause when complete	-w	<PauseAtEnd>	Pause and wait for a key to be pressed when the script has finished executing. Use this option only when run interactively (from the command line).
Display commands	-y	<DisplayCommands >	Display the script commands to the console as they are executed.
Script content	n/a	<ScriptContent>	When omitted an external script file must be used and the script filename setting specified.

Note that the output filename, script filename and variable filename may contain date and time codes that allow the current date and time to be automatically included in them. These codes include #yyyy, #yy, #mmm, #mm, #ddd, #dd, #hh, #nn, #ss and #zzz, representing the current year (4 or 2 digit), month (abbr. name or 2 digit), day (abbr. name or 2 digit), hour, minute, second and millisecond.

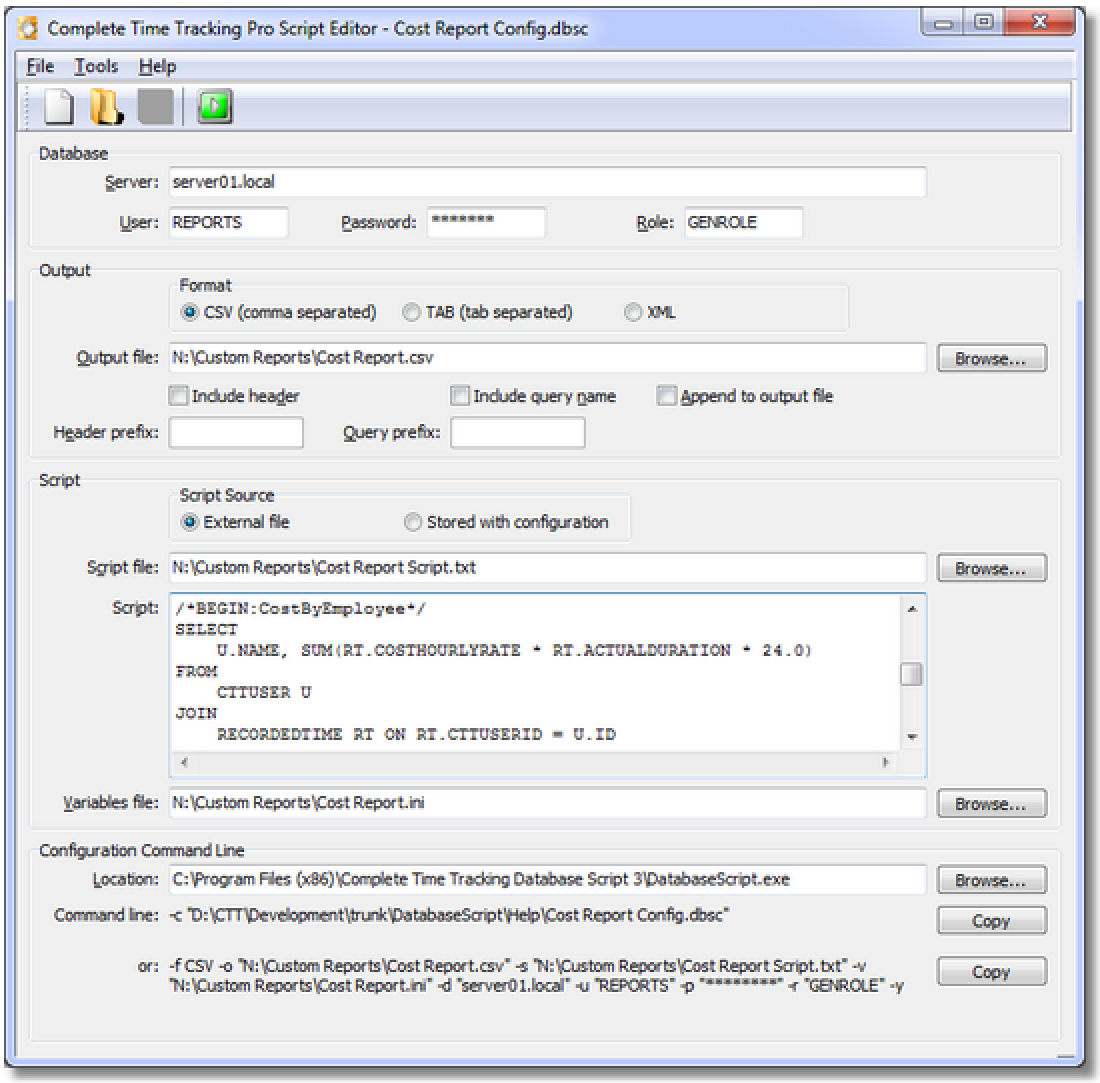
Script Commands

A number of commands can be used within scripts such as SQL statements (SELECT, INSERT, UPDATE, DELETE), output statements, conditional execution, and the use of variables that can be saved between runs allowing the following run to resume where the last run finished.

See also:
Configuration Editor
Using Command Line Options
Configuration File
Script Language

4.2 Configuration Editor

The Database Script configuration editor is an interactive application that allows you to easily create and modify configuration files and command line options for the Database Script command line tool.


Complete Time Tracking Pro Script Editor - Cost Report Config.dbsc
File Tools Help
Browse...
Copy

Database
Server: server01.local
User: REPORTS Password: ******* Role: GENROLE

Output
Format: [✓ CSV (comma separated)] TAB (tab separated) XML
Output file: N:\Custom Reports\Cost Report.csv
☐ Include header ☐ Include query name ☐ Append to output file
Header prefix: Query prefix:

Script
Script Source: [✓ External file] Stored with configuration
Script file: N:\Custom Reports\Cost Report Script.txt
Script:

```
/*BEGIN:CostByEmployee*/
SELECT
    U.NAME, SUM(RT.COSTHOURLYRATE * RT.ACTUALDURATION * 24.0)
FROM
    CTTUSER U
JOIN
    RECORDEDTIME RT ON RT.CTTUSERID = U.ID
```

Variables file: N:\Custom Reports\Cost Report.ini

Configuration Command Line
Location: C:\Program Files (x86)\Complete Time Tracking Database Script 3\DatabaseScript.exe
Command line:

```
-c "D:\CTT\Development\trunk\DatabaseScript\Help\Cost Report Config.dbsc"
```

or:

```
-f CSV -o "N:\Custom Reports\Cost Report.csv" -s "N:\Custom Reports\Cost Report Script.txt" -v
"N:\Custom Reports\Cost Report.ini" -d "server01.local" -u "REPORTS" -p "*********" -r "GENROLE" -y
```

Database Script Configuration Editor

Loading and Saving Scripts

The editor can be used to save, load and edit configuration files. Configuration files can be saved in plain-text or encrypted password protected format. The configuration file filename extension determines the format:

- **dbsc**: Plain text
- **dbscx**: Encrypted and password protected

Select the appropriate file type or enter the appropriate extension when saving the file. When saving or loading an encrypted configuration file you must enter the password that protects the file.

Testing Scripts

The editor has a test function that can call the Database Script command line tool if the configurable settings are saved to a configuration file. This can be used to test the database connection and output file formatting settings and to test the script itself.

Warning: If your script modifies the database in any way, such as through SQL INSERT, UPDATE or DELETE commands then you should be extremely careful designing and testing the script. You must also conform to the Complete Time Tracking direct access requirements.

See also:
Configuration
Using Command Line Options
Configuration File
Script Language

4.3 Using Command Line Options

Running the Database Script tool requires specifying several configuration options. Most configuration options can be specified on the command line or alternatively stored in a configuration file in which case the filename of the configuration file must be specified on the command line.

An interactive editor is provided to simplify the process of creating the command line options to be used. When configuration options are selected in the editor or a previously saved configuration file is loaded the formatted command line parameters to use when calling the Database Script tool are displayed and can be copied to the Windows clipboard.

Specifying Options on the Command Line

Each configuration option has an equivalent command line option. These are described in the Configuration section. Command line options can be specified in any order.

Example Using Only Command Line Options

```
DatabaseScript.exe -f CSV -o "Output.csv" -s "Script.txt" -d
"server01" -u "GENUSER" -p "password" -r "role"
```

When command line options are used the database script commands must be stored in a script file and specified using the -s command line option.

Specifying Options in a Configuration File

The configuration file is specified using the -c command line option. This is the only command line option required.

Example Using a Configuration File

```
DatabaseScript.exe -c "filename.dbsc"
```

When a configuration file is used the script commands can either be stored within the configuration file or a separate script file (in which case the configuration file specifies the script filename).

Scheduled Task

The Database Script tool can be run as a Windows scheduled task. Create a new scheduled task and specify the Database Script tool program name DatabaseScript.exe and command line options as described in the previous sections. Variables can be used to save state between executions and provide resume/continue functionality.

See also:

Configuration
Configuration Editor
Configuration File

4.4 Configuration File

Running the Database Script tool requires specifying several configuration options. Most configuration options can be specified on the command line or alternatively stored in a configuration file in which case the filename of the configuration file must be specified on the command line. The configuration file uses an XML file format and typically has the filename extension *.dbsc*.

An interactive editor is provided to simplify the process of creating the configuration file to be used.

Specifying Options in a Configuration File

The configuration file is specified using the -c command line option. This is the only command line option required. Run the tool as follows:

```
DatabaseScript.exe -c "filename.dbsc"
```

Where "filename" is the name of the configuration file, including the drive and path if necessary.

When a configuration file is used the database script commands can either be stored within the configuration file itself or in a separate script file (in which case the configuration file specifies the script filename).

Example configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<Config>
  <ScriptFileName>ExampleScript.txt</ScriptFileName>
  <OutputFileName>ExampleScriptOutput.csv</OutputFileName>
  <VariableFileName>ExampleScriptVariables.ini</VariableFileName>
  <AppendToOutputFile>0</AppendToOutputFile>
  <OutputFormat>CSV</OutputFormat>
  <IncludeHeader>1</IncludeHeader>
  <HeaderPrefix></HeaderPrefix>
  <IncludeQueryName>1</IncludeQueryName>
  <QueryNamePrefix></QueryNamePrefix>
  <Database>localhost</Database>
  <DatabaseUser>xxxxxxxx</DatabaseUser>
  <DatabasePassword>xxxxxxxx</DatabasePassword>
  <DatabaseRole>xxxxxxxx</DatabaseRole>
  <ScriptContent></ScriptContent>
  <DisplayCommands>0</DisplayCommands>
  <PauseAtEnd>0</PauseAtEnd>
</Config>
```

The values in the configuration file are described in detail in the Configuration section. The values are summarized in the following table.

Configuration File Option	Value Type	Description	Example
ScriptFileName	Text	Valid filename without quotes.	ExampleScript.txt
OutputFileName	Text	Valid filename without quotes. If omitted or blank the output will be sent to the console.	ExampleScriptOutput.csv
VariableFileName	Text	Valid filename without quotes.	ExampleScriptVariables.ini

AppendToOutputFile	Boolean	0 (false) or 1 (true)	0
OutputFormat	Text	Must have the value CSV, TAB or XML.	CSV
IncludeHeader	Boolean	0 (false) or 1 (true)	1
HeaderPrefix	Text	User defined.	hdr:
IncludeQueryName	Boolean	0 (false) or 1 (true)	1
QueryNamePrefix	Text	User defined.	qry:
Database	Text	Host name or IP address of server computer with optional database name and optional TCP port.	localhost
DatabaseUser	Text	The database user (not a Complete Time Tracking user) to connect as. The user determines the access privileges. Users are configured using the Complete Time Tracking Server Manager tool.	GENUSER
DatabasePassword	Text	Password for database user.	password
DatabaseRole	Text	Role for the database user.	GENROLE
ScriptContent	Text	The database script commands when an external script file is not used.	See Database Script
DisplayCommands	Boolean	Display the script commands to the console as they are executed.	0
PauseAtEnd	Boolean	Pause and wait for a key to be pressed after all script commands have been executed.	0

See also:
Configuration
Configuration Editor
Configuration File

4.5 Direct Access Requirements

Warning: You MUST read and fully understand this section if you use SQL Statement commands. The section contains important information on how to **identify valid data** in the database and **if your script modifies the database** in any way you should be extremely careful designing and testing the script and must conform to the **specified rules**.

Access Privileges

In order to access the database a valid database username, password and role must be specified. Database users can be managed from the Complete Time Tracking Server Manager tool which is installed on the server computer. The following roles are available and must match the appropriate access setting specified when the database user was created:

GENROLE: Limited read only access. Not all tables or selectable stored procedures can be accessed and views are provided for accessing a limited set of data from select tables.
ADVROLE: Full read only access. All tables and selectable stored procedures can be accessed.
SYSROLE: Read-write access. Rows can be selected, inserted, updated and deleted (in cases where row deletion is valid).

If upon running the script you see the following error you need to ensure that the correct role was specified:
no permission for read/select access to TABLE CTTSYSTEM

Important Database Design Features

Currently all data stored in the database is 8-bit ASCII without regard to specific character sets. Unicode is planned but not currently supported.

Rows in the following tables are versioned to provide the necessary two-way merge replication information for the offline time tracking feature:

CTTUSER
USERSECURITYACTION
CATEGORY
RECORDEDTIME

Versioning has three implications:

1. Rows in these tables are **never deleted**. To maintain the necessary version history and synchronization capability for the offline time tracking feature deleted rows in these tables are retained and marked with a deleted status. They are not actually removed from the database. Each of these tables has a ROWSTATUS column with one of two single character values:

   'A' - Active. **Only these rows** should be included in reporting queries.
   'D' - Deleted. These rows should be **omitted** from reporting queries.

2. These tables have **CREATEROWVERSION** and **ROWVERSION** columns, storing an integral version number. CREATEROWVERSION and ROWVERSION store the version number at the time that the row was inserted into the database. A single version number generator is shared by all tables. The version number is incremented for each set of changes made to the database and the ROWVERSION column is updated with the new version number whenever the row is updated (including an update marking the row as deleted as described above). CREATEROWVERSION does not change.

3. References to changes in these tables and additional information are recorded in the **VERSIONHISTORY** table.

Unique Identifiers

Other than the CTTSYSTEM table each table has a column named ID storing an integral value (number). This is used as the primary key and uniquely identifies each row. When inserting rows the unique ID generator for the table must be used to obtain a new ID. See Inserting and Updating Data.

Linking to External Systems

The CTTUSER and CATEGORY tables include one and two external reference columns respectively. These can be set to unique identifiers used in external systems to provide a method of linking the data in the two systems.

Retrieving Data

When querying data from versioned tables only rows with ROWSTATUS set to 'A' should be included. This will omit any deleted data.

Foreign keys exist in tables to reference related rows in other tables. These can be used to join data from multiple tables into a single result set. For example, rows in RECORDEDTIME have foreign keys referencing the user in the CTTUSER table and category in the CATEGORY table.

Example Query Including Active Rows Only

```
SELECT NAME FROM CTTUSER
WHERE ROWSTATUS='A'
ORDER BY USERNAME;
```

Time Entries

The RECORDEDTIME table stores time entries for all users. There is a foreign key to the user and category.

Note that the duration columns are specified in days. To convert to hours multiply by the number of hours in a day, 24.0. For example, a duration of 0.094162 days = 2.256 hours (0.094 x 24.0) which equals 2 hours 15 minutes 35.6 seconds. The DURATION column stores the duration to use in reports. It is set to the value of SETDURATION (a value optionally set manually by a user for rounding purposes) if not NULL else ACTUALDURATION (end date-time minus start date-time).

Categories

The CATEGORY table is self-referencing. A parent-child relationship exists between categories to provide hierarchical categorization. The hierarchy can be *any number* of levels deep. The selectable stored procedure SP_CATEGORYTREE is provided to retrieve the hierarchical structure necessary in reports. You can use it as follows.

```
SELECT *
FROM SP_CATEGORYTREE(
    NULL, NULL, NULL, NULL, NULL, [A], 'Y', [B], [C], 'Y')
```

Where:

- [A] is the depth of the tree you want such as 3 (sub-categories below the depth are subsumed into the parent) or NULL for the entire tree.
- [B] is the order of categories and is 'D' for description or 'C' for category code.
- [C] is the visibility area restriction. 'S' = selection, 'R' = reports, NULL = all categories.
- The other initially NULL columns are used internally to track recursion.

Categories can have visibility restricted to certain areas. Four columns VISIBILITYSELECT, VISIBILITYCONFIG, VISIBILITYEDITTIME, and VISIBILITYREPORTS determine whether the category is displayed for selection on the Main Tracking window, in the Category Configuration window, Edit Time Entries window, and Reports window respectively. When set to 'Y' the category is visible.

Categories can have estimated durations in ORIGINALESTIMATEDURATION and CURRENTESTIMATEDURATION. These are stored in the units based on the value in the ESTIMATETIMEUNIT column. The time units are 'H' hours, 'D' days, 'W' weeks, 'M' months. Two columns, ORIGINALESTIMATEDURATIONDAYS and CURRENTESTIMATEDURATIONDAYS, contain the calculated estimated durations in the common time unit of days for convenience.

Example Queries Using Categories

```
SELECT
  ID, PARENTID, CATEGORYLEVEL, DESCRIPTION, NOTES, HOURLYRATE,
  PERCENTCOMPLETE
FROM
  SP_CATEGORYTREE(NULL, NULL, NULL, NULL, NULL, NULL, 'Y', 'D', NULL,
'Y')

SELECT
  C.FULLDESCRIPTION, RT.STARTDATETIME, RT.ENDDATETIME, RT.DURATION *
24.0
FROM
  SP_CATEGORYTREE(NULL, NULL, NULL, NULL, NULL, NULL, 'Y', 'D', NULL,
'Y') C
LEFT JOIN
  RECORDEDTIME RT ON RT.CATEGORYID = C.ID
WHERE
  RT.STARTDATETIME >= '2009-12-25' AND RT.ROWVERSION = 'A'
ORDER BY
  RT.STARTDATETIME
```

Making Changes to Data and Schema

Consider that the database has been specifically designed to meet the functional requirements of Complete Time Tracking. Strict rules must be followed when inserting, updating or deleting data, and for any changes made to the database schema. Technical support cannot be provided if changes made to the database interfere with the operation of the product.

The following list contains a few general rules. Specific updating rules are provided in the following paragraphs.

- You **must not alter** any existing domain, stored procedure, trigger, index, generator or view or the structure of any existing table.
- The content of the CTTSYSTEM and GLOBALOPTION tables **must not to be changed** in any way.
- The VERSIONHISTORY, CTTTAG, CTTUSERTAG, CATEGORYTAG and RECORDEDTIMETAG tables are maintained automatically and **changes must not be made to the tables directly**.
- If the audit trail feature is enabled (a system level feature) the details of changes made to the versioned tables are stored in the AUDIT table to facilitate searching and reporting. This table is **maintained automatically** and other than deleting old audit data changes should not be made to the AUDIT table directly.
- Any custom database objects created by you such as stored procedures, generators, tables, or triggers on those tables must be **prefixed with 'CUSTOM'** to avoid future name clashes.
- You **must not create any dependencies** between custom database objects and the existing application database objects. For example you must not create any foreign keys to existing tables, triggers on existing tables, or stored procedures, triggers or views that reference existing tables. Consider that an automated script is used in Complete Time Tracking to make changes to the schema and data for new product versions. Certain automated changes will fail if unexpected dependencies exists on the objects being changed, resulting in an incomplete upgrade that cannot be completed until the dependencies are removed.

<u>Inserting and Updating Data</u>

Row Identifiers

When inserting a row you must obtain a new unique ID for the row using the appropriate generator for the table. Generator names are prefixed with "G_". You should always increment the generator value by 1. For example:

```
INSERT INTO CTTUSER
  (ID, USERNAME, ...)
VALUES
  (GEN_ID(G_CTTUSER, 1), 'UserName', ...)
```

There are triggers on the tables to assign an ID automatically but it is best to obtain a specific value (as is required when creating dependent rows in other tables which need to use it in a foreign key).

Versioning

Each **set of changes** to the versioned tables requires a unique ROWVERSION which you can obtain from the G_ROWVERSION generator. You may (and are encouraged to) use the same ROWVERSION for all changes in a change-set in a single transaction. This improves the speed of the synchronization process with offline time tracking. Obtain a new value for the next transaction. For INSERT's also set CREATEROWVERSION to the same ROWVERSION value.

For example:

```
UPDATE CTTUSER
SET NAME = 'Fred Smith',
    ROWVERSION = GEN_ID(G_ROWVERSION, 1)
```

```
WHERE ID = 123 AND ROWSTATUS = 'A'
```

Tags

Where the table supports tags (CTTUSER, CATEGORY, RECORDEDTIME) the TAGNAMES column should be set to the comma-separated tag values. A trigger and stored procedures will then automatically update the CTTTAG with each unique tag value and the referencing CTTUSERTAG, CATEGORYTAG and RECORDEDTIMETAG tables. The tag tables should not be modified directly.

Deleting Data

Do not delete rows from the versioned tables. Deleted rows must be retained to keep version history used in offline time tracking synchronization. Instead of deleting rows in these tables update them and set ROWSTATUS to 'D'. As with all other changes you **must also update the ROWVERSION** to a new value obtained from the G_ROWVERSION generator (see Inserting and Updating Data).

See also:
SQL Statement Command

4.6 Database Script

The database script is a sequence of commands that perform various actions, such as retrieving or updating data. Scripts can be used to extract custom data that is otherwise not possible through the pre-defined reports in the Complete Time Tracking user interface. This data can be used in other reporting tools or applications such as Microsoft Excel. Scripts could also be used to add, modify or delete users, categories, time entries and other data programmatically.

Scripts support several commands such as SQL statements, output statements, conditional execution (running a command if a certain condition is true), and the use of variables that can be saved between runs allowing the following run to resume where the last run finished.

Warning: You MUST read and fully understand the direct access requirements section which contains important information on how to identify valid data in the database and if your script modifies the database in any way, such as through SQL INSERT, UPDATE or DELETE commands then you should be extremely careful designing and testing the script and must conform to the specified rules.

Example Script

A simple example script containing five commands follows:

```
/*NAME:Users*/
SELECT USERNAME FROM CTTUSER
WHERE ROWSTATUS = 'A'
ORDER BY USERNAME;

SETVAR VARIABLE1 TO 123;

OUTPUTLINE "The value is :VARIABLE1";

IF (:VARIABLE1 > 5) AND (:VARIABLE1 < 200) THEN
  SELECT USERNAME FROM CTTUSER
  WHERE ROWSTATUS = 'A' AND NAME LIKE '%JOHN%'
  ORDER BY USERNAME;

SAVEVAR VARIABLE1 TO FIXEDVALUE;
```

The previous script writes a list of usernames to the output file, writes the text *The value is 123* to the output file, writes a list of usernames to the output file for users with the name containing the text JOHN, and finally saves the value *123* to the variables file with a variable named FIXEDVALUE.

See also:
Direct Access Requirements
Script Language
Variables
SQL Statement Command
OUTPUTLINE Command
SETVAR Command
SAVEVAR Command
LOADVAR Command
CLEARVAR Command
DELETEVAR Command
IF Command

4.6.1 Script Language

Scripts contain one or more commands in a defined format. Specific keywords and expressions form the script language.

Script Commands

The script commands are as follows:

SQL Statement: SELECT to extract data from the database and write it to the output file, INSERT, UPDATE and DELETE to change data in the database.
OUTPUTLINE: Write the given text or variable value to a new line in the output file.
SETVAR: Set the value of a variable.
SAVEVAR: Save a variable to the variables file for use in subsequent script runs.
LOADVAR: Load a previously saved variable from the variables file.
CLEARVAR: Remove a variable (clear it) from the current run.
DELETEVAR: Delete a saved variable from the variables file.
IF: Conditional execution of another command.

Command Format

Each command can be written over one or more lines and is terminated with a semi-colon character ';'. For example:

```
SELECT USERNAME FROM CTTUSER
WHERE ROWSTATUS = 'A'
ORDER BY USERNAME;
```

SQL Statement commands containing DDL (such as the creation of temporary tables or stored procedures) must be enclosed in an explicit section between /*BEGIN*/ and /*END*/ markers as follows:

```
/*BEGIN*/
first command line
second command line
...
last command line;
/*END*/
```

Comments

Comments can be added to the script. Single line comments begin with a double forward slash // and cause the remainder of the line to be ignored. Multi-line comments can be enclosed between the /* and */ tokens. For example:

```
// This is a single line comment
/* This is another comment */
/* This comment
spans
multiple lines */
```

Note: Comments are not allowed on the /*BEGIN*/, /*END*/, /*NOCOMMIT*/ or /*NAME*/ marker lines.

Variables

Scripts can make use of variables to store and use named data values.

See also:
Variables
SQL Statement Command
OUTPUTLINE Command
SETVAR Command
SAVEVAR Command
LOADVAR Command
CLEARVAR Command
DELETEVAR Command
IF Command

4.6.2 Variables

The database script supports the use of variables. Variables provide the ability to store and retrieve named data for use in script commands. Variables can be saved to a variables file for use in a subsequent run of a database script. Variables have a user defined name and must consist solely of alphanumeric characters or the underscore character (a-z,A-Z,0-9,_). Variable names are case sensitive (MYVARIABLE is distinct from MyVariable for example) and may not contain spaces.

Valid example variable names:
MYVARIABLE
MyVariable
My_Variable
Variable1000A

Invalid example variable names:
My Variable
My-Variable
VariableA!

Variables are used in script commands by prefixing them with the colon ':' character. When a variable is used it is substituted with the variable's literal value. Special handling of variables in SQL Statement commands allows the variable values to be directly accessed as SQL parameters. Text variables in SQL Statement commands do not need to be enclosed in quotes. Date and time values will be interpreted using the Windows regional settings. Exact variable substitution with its literal value can be performed in SQL statements using a double colon prefix :: if desired. This direct access also alleviates the need to escape or otherwise format values that may invalidate the SQL statement.

For example if the variable named VARIABLE1 had the value *A Sample Message* then the following script command would write the text *This is A Sample Message* to the output file.

```
OUTPUTLINE "This is :VARIABLE1";
```

The following example shows variable use in a SQL Statement command. If the variable named NEWNAME has the value *Sam Gregory* then the following script command would set the descriptive name of the user with username samg to *Sam Gregory*. Note that the variable is not enclosed in quotes as would normally.

```
UPDATE CTTUSER
SET NAME = :NEWNAME
WHERE USERNAME = 'samg'
AND ROWSTATUS = 'A';
```

SQL Statement Examples
The following example uses direct variable access in SQL Statement commands. Here the variable STARTDATE contains the value 01/01/2010, a valid date format on the computer that the script is being run on. Note that the variable is not being treated as text and therefore does not require surrounding quotes.

```
SELECT DESCRIPTION FROM CATEGORY
WHERE TARGETCOMPLETIONDATE >= :STARTDATE
AND ROWSTATUS = 'A';
```

If the variable STARTDATE instead contained the value 2010-01-01, a valid SQL date format but not a valid Windows date format, then literal text replacement must be used instead of direct access. Note that the variable is prefixed with the double colon for literal replacement and is surrounded by quotes to indicate a SQL text value:

```
SELECT DESCRIPTION FROM CATEGORY
WHERE TARGETCOMPLETIONDATE >= '::STARTDATE'
AND ROWSTATUS = 'A';
```

Check the use of variables carefully. If a variable does not exist then no value substitution takes place. In the case of SQL Statement commands an error will be produced. In the case of all other commands the variable name will be treated as text and likely produce an error or have an undesired result.

IF Command
When a variable is used in the conditional expression of an IF command (the part that is evaluated as either true or false) the variable name is substituted with the value of the variable. If you are using the variable to store general text (not a number) then you must enclose the variable name with quotes. For example, if variable VARIABLE1 has the value *123.456* you can use it as-is to treat it as a number as in the expression *IF :VARIABLE1 > 100*, or enclose it in quotes to treat it as text as in the expression *IF Length(":VARIABLE1") > 5*.

See also:
Script Language
SETVAR Command
SAVEVAR Command
LOADVAR Command
CLEARVAR Command
DELETEVAR Command

4.6.3 SQL Statement Command

SQL (sequel) is a generic database language used by the database system in Complete Time Tracking and universally in database applications throughout the world. SQL is used to retrieve data from and update the data and metadata in databases. The SQL Statement commands include SELECT, INSERT, UPDATE and DELETE and conform to the SQL-92 specification.

Contact Complete Time Tracking support to obtain database schema details which list the tables and columns in the database.

Warning: You MUST read and fully understand the direct access requirements section if you used SQL Statement commands. The section contains important information on how to **identify valid data** in the database and **if your script modifies the database** in any way you should be extremely careful designing and testing the script and must conform to the **specified rules**.

See the section on Variables for specific details of how variables are interpreted in SQL Statement commands.

SELECT Command

The SELECT query command allows you to retrieve data from the database. The data is automatically written to the output file in the format specified.

For example:

```
SELECT NAME FROM CTTUSER
WHERE ROWSTATUS = 'A'
ORDER BY USERNAME;
```

Where multiple SELECT commands are included in the same database script multiple sets of data will be written to the output file in the order in which the SELECT commands appear in the script. The Query Name option (see the following sections) can be used to identify the data from each SELECT command.

Append to Output File

If the Append to Output File option is enabled and the output file already exists, possibly from a previous run of the script, the data for the current run will be appended to the end of the existing file; that is the existing file contents will not be overwritten.

Header

When the Include Header configuration option is enabled the name of each column specified in the SELECT command is written to the output file prior to the data rows.

Header Prefix

This option is used with the CSV and TAB output file formats. If a value is provided for the Header Prefix option the header line in the output file is prefixed with the configured prefix. This makes it possible to identify the header programmatically, particularly when multiple SELECT commands are included in the same script or the Append to Output File configuration option is enabled.

Query Name

An optional user-defined query name may be specified for SELECT SQL commands using the /*NAME:<name>*/ marker. When the Include Query Name configuration option is enabled the specified query name is written to the output file prior to the header or data rows. This makes it possible to identify the output from each SELECT command, particularly when multiple SELECT commands are included in the same script or the Append to Output File configuration option is enabled.

For example:

```
/*NAME:QueryName*/
SELECT ... ;
```

Query Name Prefix

This option is used with the CSV and TAB output file formats. When the Include Query Name configuration option is enabled and a value is provided for the Query Name Prefix option the query name specified in the NAME marker is prefixed with the configured query name prefix. This makes it possible to identify the query name for this SELECT command programmatically.

Example SELECT Command and Output Data

The following example assumes that the Include Header and Include Query Name options are enabled and the Header Prefix is set to "HeaderPrefix:" and Query Name Prefix set to "QueryNamePrefix:".

Example script command:

```
/*NAME:Users*/
SELECT USERNAME, NAME, NOTES FROM CTTUSER
WHERE ROWSTATUS = 'A'
ORDER BY USERNAME;
```

Example output:

Assuming that there were two active users in the database the data would be written to the output file as follows:

CSV:
```
QueryNamePrefix:Users
HeaderPrefix:"USERNAME","NAME","NOTES"
"Admin","Administrator","The system administrator"
"john","John Smith","Department Manager, ext 303"
"zelda","Zelda Robinson","Service Technician"
```

TAB:
```
QueryNamePrefix:Users
HeaderPrefix:USERNAME   NAME  NOTES
Admin Administrator     The system administrator
john  John Smith  Department Manager, ext 303
zelda Zelda Robinson    Service Technician
```

XML:
```
<?xml version="1.0" encoding="utf-8"?>
<Data>
  <DataSet name="Users">
    <Header>
      <FieldName index=0>USERNAME</FieldName>
      <FieldName index=1>NAME</FieldName>
      <FieldName index=2>NOTES</FieldName>
    </Header>
    <Rows>
      <Row>
        <USERNAME>Admin</USERNAME>
        <NAME>Administrator</NAME>
        <NOTES>The system administrator</NOTES>
      </Row>
      <Row>
        <USERNAME>john</USERNAME>
        <NAME>John Smith</NAME>
        <NOTES>Department Manager, ext 303</NOTES>
      </Row>
      <Row>
        <USERNAME>zelda</USERNAME>
        <NAME>Zelda Robinson</NAME>
        <NOTES>Service Technician</NOTES>
      </Row>
    </Rows>
  </DataSet>
</Data>
```

Using SELECT Commands with SETVAR

A singleton SELECT command that returns a single value can also be used with the SETVAR command to store the value in a variable. For example:

```
SETVAR VariableName TO
  SELECT NAME
  FROM CTTUSER
  WHERE USERNAME = 'Admin';
```

Restricted Data Access

Database users are assigned one of three roles:

- GENROLE: Limited read.
- ADVROLE: Full read.
- SYSROLE: Read write.

These roles determine the read and write data access for the user. Users with limited read functionality, GENROLE, will be unable to retrieve data directly from many tables. The following views are provided to allow the retrieval of a limited set of data from the CTTUSER, CATEGORY and RECORDEDTIME tables:

- V_USERGEN
- V_CATEGORYGEN
- V_RECORDEDTIMEGEN

INSERT, UPDATE and DELETE Commands

The INSERT, UPDATE and DELETE SQL commands modify data in the database, adding new rows, changing rows and deleting rows respectively. They must be used with great caution as there are particular rules that must be followed when using them.

General Stored Procedures

The following stored procedures are available for general use:

- SP_CATEGORYTREE: Returns the category hierarchy from a given parent category (or all categories) in description or code order to the depth requested.
- SP_CATEGORYTREE_BYCODE: A convenient way to use SP_CATEGORYTREE without the need to provide input parameters. Returns the entire category hierarchy in code order with descriptions up to 5 levels deep.
- SP_CATEGORYTREE_BYDESC: A convenient way to use SP_CATEGORYTREE without the need to provide input parameters. Returns the entire category hierarchy in description order with descriptions up to 5 levels deep.
- SP_CATEGORYSTRUCTURE: Fast category hierarchy (ID, PARENTID) without regard to order of categories within a given parent.
- SP_CATEGORYTREE_FILTERED: Returns the hierarchy of categories given a specific filter (WHERE clause).
- SP_WEEKLYTIMESHEETREPORT: Total duration data for a 7-day period by category.
- SP_SPLITTEXT: Split the given delimited text into individual values.
- U_AUDITENABLED: Returns 'Y' if the audit trail feature is enabled, else 'N'.
- U_CRLF: Returns a Windows carriage-return/linefeed character pair.
- U_CURRENTDATETIME: Returnes the current date and time of the server.
- U_TIME_UNIT_TO_DURATION: Convert a duration in a given time unit to a duration in days.
- U_DB_VERSION: Returns the current database version.

Additional SQL Functions

In addition to the standard SQL functions such as SUM, MIN and MAX the following external functions are available for use in SQL statements:

ASCII_CHAR, ASCII_VAL, CEILING, DIV, FLOOR, LOWER, LPAD, LTRIM, MOD, RAND, RPAD, RTRIM, STRLEN, SUBSTR, SUBSTRLEN.

See also:
Direct Access Requirements
Database Script

4.6.4 OUTPUTLINE Command

The OUTPUTLINE command simply writes the given text to a new line in the output file. It is useful for writing calculated values, changing the format of output data from a SELECT SQL command, or adding data that can be programmatically used by a program that reads the output file.

Note: If the output file format is XML then writing arbitrary text to the output file can make the XML unreadable by automated programs.

Example Commands

```
OUTPUTLINE "Look at the following data:";

OUTPUTLINE :VARIABLE1;

OUTPUTLINE "The values are :FIXEDVARIABLE1, :FIXEDVARIABLE2";
```

See also:
Database Script

4.6.5 SETVAR Command

The SETVAR command initializes or sets the value of a variable. A variable is a named item that can store data for use in other areas of the script. SETVAR can be used to set a value directly, use other previously set variables, or use a value from a SELECT SQL statement.

The value of a variable can be cleared using the CLEARVAR command.

Examples Using a Direct Value

```
SETVAR VARIABLE1 TO 123;

SETVAR VARIABLE1 TO "Some text";
```

Example Using Other Variables

```
SETVAR VARIABLE1 TO ":THEUSERNAME's name is :THENAME";
```

Example Using SQL SELECT Statement

```
SETVAR AdministratorName TO
  SELECT NAME
  FROM CTTUSER
  WHERE USERNAME = 'Admin' AND ROWSTATUS = 'A';
```

Example Using SQL SELECT Statement to Perform Simple Calculations

All SQL SELECT statements require a FROM clause. When the statement does not retrieve data from tables within the database you must use rdb$database (not case sensitive) in the FROM clause.

```
SETVAR NextValue TO
  SELECT :NextValue + 1
  FROM rdb$database;
```

See also:
Database Script
SQL Statement
CLEARVAR Command

4.6.6 SAVEVAR Command

The SAVEVAR command stores the value of a variable to the saved variables file for use in subsequent script runs. By default the value is saved using the same name as the variable name. Alternatively a different name can be provided for the saved variable. If the variable already exists in the saved variables file it will be overwritten with the new value.The value of the variable can be retrieved using the LOADVAR command. The saved variable can be deleted from the variables file using the DELETEVAR command.

Example Using Default Saved Variable Name

```
SAVEVAR VARIABLE1;
```

Assuming the variable named VARIABLE1 has the value *Example Value* the contents of the variables file after the SAVEVAR command will be as follows:

```
[VARIABLES]
VARIABLE1=Example Value
```

Example Using Alternative Saved Variable Name

```
SAVEVAR VARIABLE1 TO AnotherName;
```

Assuming the variable named VARIABLE1 now has the value *Another Value* and both commands above were run the contents of the variables file after the second SAVEVAR command will be as follows:

```
[VARIABLES]
VARIABLE1=Example Value
AnotherName=Another Value
```

See also:
Database Script
LOADVAR Command
DELETEVAR Command

4.6.7 LOADVAR Command

The LOADVAR command loads the value of a variable from the saved variables file previously saved using the SAVEVAR command. By default the value is loaded using the same name as the variable name. Alternatively a different name can be provided for the saved variable to load from. The saved variable can be deleted from the variables file using the DELETEVAR command.

Example Using Default Saved Variable Name

```
LOADVAR VARIABLE1;
```

Assuming the saved variables file has the following content the variable named VARIABLE1 would be loaded with the value *Example Value*:

```
[VARIABLES]
VARIABLE1=Example Value
```

Example Using Alternative Saved Variable Name

```
LOADVAR VARIABLE1 FROM AnotherName;
```

Assuming the saved variables file has the following content the variable named VARIABLE1 would be loaded with the value *Another Value*:

```
[VARIABLES]
VARIABLE1=Example Value
AnotherName=Another Value
```

See also:
Database Script
SAVEVAR Command
DELETEVAR Command

4.6.8 CLEARVAR Command

The CLEARVAR command clears the value previously assigned to a variable using the SETVAR command or LOADVAR command. It effectively removes the variable from the current run. If the variable was previously saved to the variables file the saved value is retained. To delete a saved variable use the DELETEVAR command.

Example

```
CLEARVAR VARIABLE1;
```

See also:
Database Script
SETVAR Command
LOADVAR Command
DELETEVAR Command

4.6.9 DELETEVAR Command

The DELETEVAR command deletes a variable from the saved variables file that was previously saved using the SAVEVAR command. It does not clear the value in a current variable set using the SETVAR command or LOADVAR command. To clear the value in a current variable use the CLEARVAR command. A variable will be assigned an empty value if set using a LOADVAR command after the variable has been deleted from the variables file.

Example

```
DELETEVAR VARIABLE1;
```

See also:

Database Script
SAVEVAR Command
SETVAR Command
LOADVAR Command
CLEARVAR Command

4.6.10 IF Command

The IF command provides conditional execution of another command. If the conditional expression specified is true then the sub-command is executed. Conditional expressions evaluate variables and values in numerical integer and floating point format. Text comparison is not currently supported directly but can be achieved indirectly by setting a variable to a numerical value using a SETVAR command in conjunction with a SQL SELECT statement performing the comparison. Values may be specified explicitly or by using variables.

Supported Conditional Expression Operators

Operators	Description
=, <>	Equal to and not equal to. True if two values are equal or not equal.
<, >	Less than, greater than. True if the first value is less than or greater than the second value.
<=, >=	Less than or equal to, greater than or equal to. True if the first value is less than or equal to, or greater than or equal to the second value.
CMP	Compares two values and evaluates to -1 if the first is less than the second, 0 if they are equal, or -1 if the first is greater than the second. For example: 5 CMP 10 would evaluate to -1.
AND, OR, XOR, NOT	Logical *and*, *or*, *exclusive or* and *not*. True if both values are true, either value is true, one but not both are true, if value is not true. See also BAND, BOR, BXOR and BNOT.
( )	Parentheses. Can be used to group sub-statements, allowing flexibility with logical comparison or mathematical calculations.
+, -, *, /	Add, subtract, multiply, divide mathematical operators. These operate on two values.
Cos, Cot, Sec, Sin, Tan	Mathematical functions. Pass a single value. For example: Cos(:VariableName)
Min, Max	Calculates the minimum or maximum of two values. For example: Min(:VariableName, 20)
Floor, Ceil, RoundTo	Floating point (decimal) number rounding. Floor rounds a number down to the nearest whole number. Ceil rounds a number up to the nearest whole number. RoundTo rounds a number to a specific number of digits. For example: Floor(1.5), Ceil(1.5), RoundTo(1.5, 1)
Random	Returns a random floating point number between 0.0 and 1.0.
StrAsTime, StrAsDate, StrAsDateTime	Converts a text value in date, time or date-time format (according to the format specified in the Windows regional settings) to a floating point (decimal) value where the integral portion is the number of days since 31/12/1899 and the decimal portion the fraction of a day. This numerical format is consistent with the Windows OLE date-time. For example, if VariableName has the value 12:00:00 then StrAsTime(":VariableName") evaluates to 0.5 (12pm is exactly half way through a day).
SavedVariableExists	Returns true if the given variable name (in quotes) exists in the saved variables file. For example:

	SavedVariableExists("VARNAME").
VariableSet	Returns true if the given variable name (in quotes) has been assigned a non-empty value. For example: VariableSet("VARNAME").
Length	Returns the character length of the given string. For example Length("ABC") equals 3, Length(":VARNAME") returns the length of the value of variable VARNAME.
IsNumber	Returns true if the given string represents an integral or floating point number such as 123 or 123.456. For example: IsNumber("123.456") returns true. IsNumber(":VARNAME") checks if the value of the variable VARNAME is a number.
BAND, BOR, BXOR, BNOT	Binary *and*, *or*, *exclusive or* and *not*. These operate on each binary digit (bit) within the integer values specified, not the number as a whole. See also AND, OR, XOR and NOT.
DIV, MOD	Integer division and remainder.
SHL, SHR	Bitwise shift left and right by the number of bits specified in the second value.

Note that in boolean (true or false) evaluation a numerical value is considered to be false if it is exactly zero (0 or 0.0) and true if it is non-zero.

Examples

```
IF :DataExists THEN
  ... ;

IF VariableSet("VARIABLE1") THEN
  ... ;

IF Length(":VARIABLE1") > 10 THEN
  ... ;

IF (:VARIABLE2 >= 5) AND (:VARIABLE2 < 200) THEN
  SELECT USERNAME FROM CTTUSER
  WHERE ROWSTATUS = 'A' AND UPPER(NAME) LIKE '%JOHN%'
  ORDER BY USERNAME;

IF (MAX(:VARIABLE1, 10) + :VARIABLE2) MOD 2 = 1 THEN
  ... ;

IF ((:FLAGS SHL 2) BAND 128) <> 0 THEN
  ... ;
```

See also:
Database Script
SETVAR Command

Part V

5 Support and Contact

We welcome contact from all users of our products. Please select the appropriate contact from the following list:

- Product and Technical Support
- Feedback, Bug Reports and Suggestions
- General Enquiries

5.1 Support

As many problems are fixed in each version of the product please ensure that you are running the latest version of Database Script. The version that you are using is displayed in the Database Script Editor about window by selecting *Help, About* from the Database Script Editor menu.

Contacting Support
If the latest program version does not fix the problem or you believe that the problem is independent of the program version then please contact us with details about the problem that you have encountered.

Support requests will be dealt with high priority. In all correspondence please include your full name and email address and phone number if possible.

Email
- **Preferred:** Select Email Support from the Help menu in Database Script Editor. This will open a new email in your default email application with the To, Subject and a template Body pre-filled. This will automatically include details about the version of Database Script and the version of Windows that you are using in the body of the email.
- **Or:** Send an email directly to support@complete-time-tracking.com

Note: Please add support@complete-time-tracking.com to the allowed list of any SPAM filter that you use to ensure that you can receive replies from us.

Voicemail and Fax
+1-800-699-0353 (US)

As we are located on the east coast of Australia and our customers are located throughout the world it is sometimes difficult to call at a suitable time and therefore email is the preferred method of contact. We evaluate this on a case by case basis.

Please mention your **name**, **country and state**, **phone number (including area code)** and **email address** in the message.

Postal Mail
Backslash Pty Ltd
PO BOX 50
Bentleigh East 3165
AUSTRALIA

See also:
General Enquiries
Feedback and Suggestions

5.2 Feedback, Bug Reports and Suggestions

Bug Reports and Program Errors

Error Report Window
If while using Database Script Editor an Error Report window is displayed please select the **Send Report** button to send the error report in an email to our technical support team. Please include a description of the problem and how it occurred and select to include a screenshot if possible.

This automated error report contains detailed technical information that will greatly assist us to locate the cause of the problem.

Sending Bug Reports
To send a general bug report please send an email to support@complete-time-tracking.com containing the following:

- The **symptoms** (including a screen shot if possible - that always helps).
- What function you were performing at the time that the error occurred.
- The **program version number** and **database version number** displayed on the About window (select *Help, About* from the menu).
- Which **version of Windows** that you are using and if you have the latest Windows updates installed.

Feedback and Suggestions
We greatly appreciate feedback and suggestions. These drive future product features and enhancements and help us to improve our service to you. Please send an email to support@complete-time-tracking.com.

Thank you in advance for your feedback. It is greatly appreciated.

See also:
Support
General Enquiries

5.3 General Enquiries

We welcome general enquiries about our products. Please see technical or product support or feedback, bug reports and suggestions if these are more appropriate options for your enquiry.

You may contact us for general enquiries using any of the methods below.

Email
- **Preferred:** Select Email Support from the Help menu in Database Script Editor. This will open a new email in your default email application with the To, Subject and a template Body pre-filled. This will automatically include details about the edition and version of Database Script and the version of Windows that you are using in the body of the email.
- **Or:** Send an email directly to support@complete-time-tracking.com

Your email will be forwarded to the appropriate team for a response.

Note: Please add support@complete-time-tracking.com to the allowed list of any SPAM filter that you use to ensure that you can receive replies from us.

Voicemail and Fax
+1-800-699-0353 (US)

As we are located on the east coast of Australia and our customers are located throughout the world it is sometimes difficult to call at a suitable time and therefore email is the preferred method of contact. We evaluate this on a case by case basis.

Please mention your **name**, **country and state**, **phone number (including area code)** and **email address** in the message.

Postal Mail
Backslash Pty Ltd
PO BOX 50
Bentleigh East 3165
AUSTRALIA

See also:
Support
Feedback and Suggestions

Index