Vitalware®
Vital Records Management

The World's Premier Vital Records Software

Vitalware Release Notes

Vitalware 2.3.01

Document Version 1

Contents

Here you will find collected together the Release Notes for Vitalware 2.3.01, alongside all documents referenced in the notes. These release notes and documents are also available on the Vitalware website.

This PDF document brings together a number of individually published documents: please note that page numbering below refers to this combined PDF document and not to the page numbers printed at the bottom of pages, as each individual document has its own internal numbering:

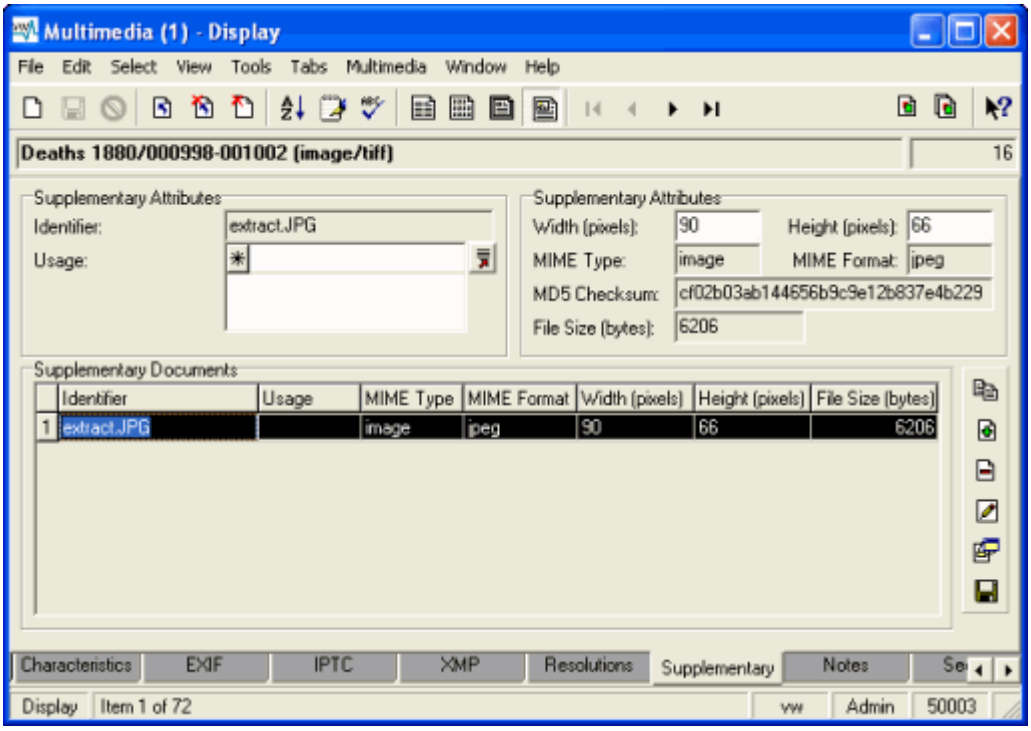
Release Notes: Vitalware 2.3.014
Release Date: 14 December 2012

Requirements

- For Windows 2000, XP, 2003, Vista, Windows 7, Windows 8
- Texpress 8.3.012 or later
- TexAPI 6.0.011 or later
- Perl 5.8.8 or later

New Features

- **Supplementary Data:** A new tab has been added to the Vitalware Multimedia Repository. The Supplementary media tab is similar in layout to the existing Resolutions tab. Supplementary media is media that is associated with the main document, but is not important enough to create a new Multimedia record. For example, you may have prepared manually a thumbnail of an image that highlights a particular part of the picture, rather than just a resize of the original. You may wish to use this image on the web as the official thumbnail. The thumbnail is not new media in its own right as it only makes sense in the context of the original image. The Supplementary media tab allows the thumbnail to be registered with the original image and flagged for use on the web:



Supplementary media is available throughout the Vitalware client, including:

  - Reporting (via the special `Supplementary_tab` column)
  - Importing (also via the `Supplementary_tab` column)
  - IMu multimedia server
  - Multimedia menus (save, print, view)

Supplementary media may be any media type, including:

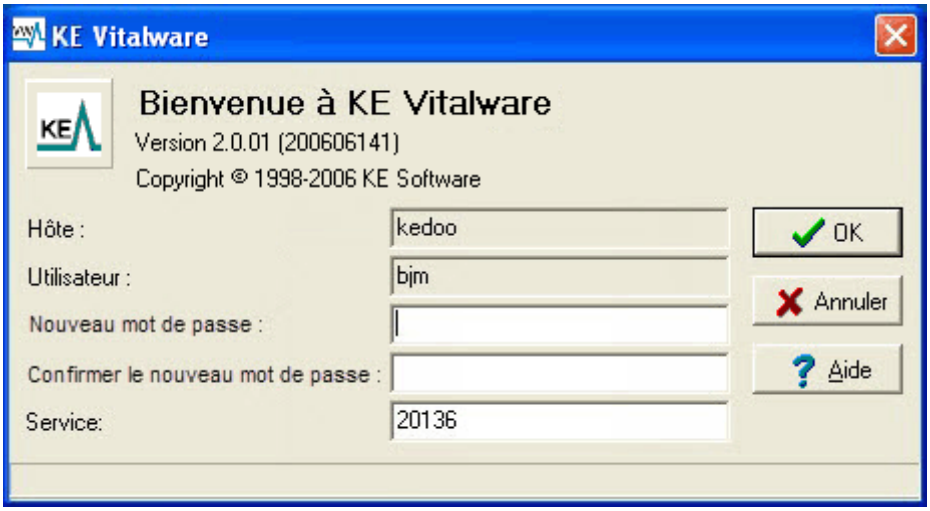
  - Images
  - Video
  - Audio
  - Electronic documents

- URLs

A complete description of the support for supplementary media can be found in the attached Supplementary Media documentation.

- **Password Management:** Password management allows users to change their login password within the Vitalware client. It also gives System Administrators the ability to implement a password security protocol within Vitalware. The following capabilities are provided to users by password management:
  - *Change Password* - Vitalware users may change their password within the Vitalware client.

  The following capabilities are provided to System Administrators:
  - *Expire Passwords* - users are forced to change their password after a specified number of days.
  - *Lock Accounts* - disable access to Vitalware for a given user account. The account may be unlocked in the future.
  - *Lock Accounts on Login Failure* - lock a user account if a specified number of unsuccessful login attempts have been made.
  - *Expire Accounts* - have a user's account locked at a specified date. The account may be unlocked in the future.
  - *Force Password Reset* - force a user to set a new password the next time they login successfully to Vitalware.
  - *Validate Passwords* - specify criteria that must be met for a new password to be acceptable (e.g. the minimum password length, the number of upper case characters required, etc.).



A complete description of the support for password management can be found in the attached Password Management documentation.

- **Dynamic Security:** Dynamic security allows security settings to be altered based on the contents of the record being displayed. The security can be set at the record level or at an individual or collection of fields level. The dynamic security settings are specified in the Vitalware Registry, allowing policy to be set on a per user, group or system-wide basis. Dynamic security consists of three parts:

  Security Update

  The Security Update Registry entry allows the record level security settings to be altered automatically based on the contents of the record being saved. For example, a record may be made read-only when the *Record Status* field contains a value of `Retired`. The format of the Registry entry is:

  `Group|`*`group`*`|Table|`*`table`*`|Security|Update|`*`column`*`|`*`value`*`|`*`settings`*

where *group* determines which Vitalware group and *table* specifies the module affected by the Registry entry. User and group default entries are also supported. When the specified *column* contains the supplied *value*, then the *settings* are applied. The *settings* allow values to be added, removed or replaced for any field in the record.

Column Access Modifier

The Column Access Modifier Registry entry allows the security settings specified by the Column Access Registry entry to be modified based on values in the existing record. The settings are evaluated after each change of value, providing immediate security modifications. The format of the Registry entry is:

`Group|`*`group`*`|Table|`*`table`*`|Column Access Modifier|`*`column`*`|`*`value`*`|`*`settings`*

where *group* determines which Vitalware group and *table* specifies the module affected by the Registry entry. User and group default entries are also supported. When the specified *column* contains the supplied *value*, then the *settings* are applied. The *settings* allow column access permissions to be added, removed or replaced for any field in the record.

Mandatory Modifier

The Mandatory Modifier Registry entry allows the mandatory settings specified by the Mandatory Registry entry to be altered based on the contents of the current record. Like the Column Access Modifier entry, the mandatory settings are evaluated after each value in changed. The format of the Registry entry is:

`Group|`*`group`*`|Table|`*`table`*`|Mandatory Modifier|`*`column`*`|`*`value`*`| `*`settings`*

where *group* determines which Vitalware group and *table* specifies the module affected by the Registry entry. User and group default entries are also supported. When the specified *column* contains the supplied *value*, then the *settings* are applied. The *settings* allow mandatory permissions to be altered for any field in the record.

The combination of Security Update, Column Access Modifier and Mandatory Modifier Registry entries allow powerful security profiles to be specified. A complete description of the support for dynamic security can be found in the attached Dynamic Security documentation.

- **Lookup List Maintenance:** The lookup list maintenance sub-system in Vitalware has been replaced with a self-maintaining system. The changes eliminate the need for the nightly or weekly reloading of the Lookup List module. Since records are no longer reloaded, the Lookup List module is now a "first class" module. Users can now add, update and delete entries in the Lookup List module directly. There are three sections that make up the modifications:

  Lookup List module

  The Lookup List module has been upgraded to a fully functional module. Records may be added, updated and deleted as in any other module in Vitalware. The following tabs have been added:

  - Notes (Attributed)
  - Multimedia
  - Security
  - Admin

  vwlutsrebuild

  The **vwlutsrebuild** program is a server-side program used to rebuild the contents of the Lookup List module based on the data stored in Vitalware. The previous version has been replaced with a version that now only applies changes to the contents of the Lookup List module. The data is no longer deleted and reloaded. The command no

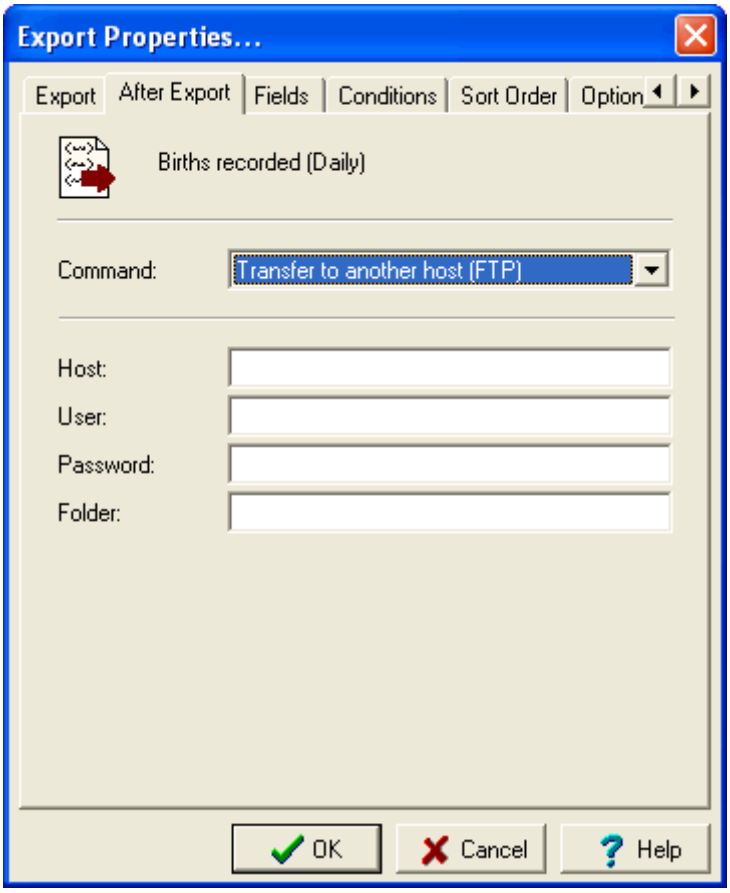
longer needs to be run regularly, but may be used to check the contents of the Lookup List module are consistent with the data stored in Vitalware.

lutserver

lutserver is a server-side program that monitors the Vitalware audit trail looking for changes in data that may affect Lookup Lists. The server ensures that new values are added, values no longer used are deleted (provided they are not permanent) and the *Used* flag is maintained.

A complete description of the new Lookup List Maintenance facility can be found in the attached Lookup List Maintenance documentation.

- **After Export Command:** The *Scheduled Export* facility introduced in Vitalware 2.1.02 has been extended to allow a post processing to occur after the export of the data is complete. The post processing can:
  - Email the export files to a list of users.
  - Email the results of the export to a list of users.
  - Copy the export files to another machine behind a secure firewall.
  - Copy the export files over the internet via a secure transfer mechanism.
  - Send an SMS to a list of telephone numbers.

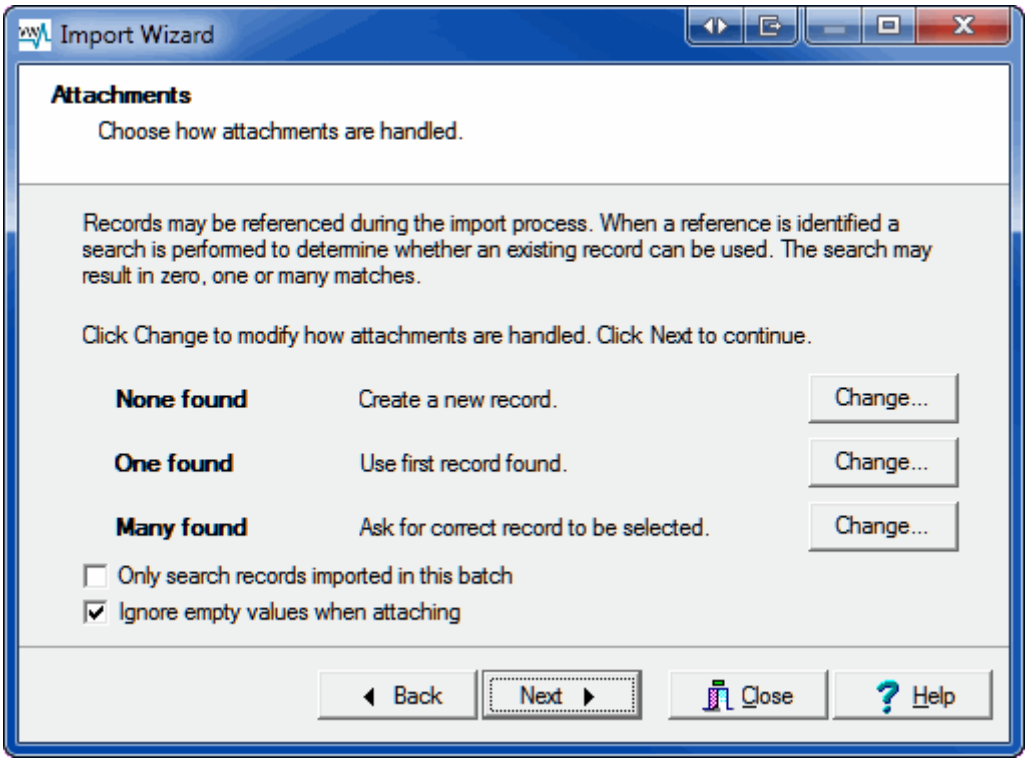


The system has been designed to be extensible, allowing new procedures to be added as required.

A complete description of the post processing facilities available can be found in the attached After Export documentation.

Improvements

- **Ignore empty values when Importing**: The Vitalware Import wizard has been extended to allow empty values to be ignored when searching for attachment records. If the option is selected, any empty field in the importing data will match all values in that field in Vitalware. This allows CSV data to be imported and attachments made where fields are left empty because a value is not known. It also allows records to match for attachments where not all values in a given hierarchy are known, or supplied, by the data source.



- **Read only Image Formats**: The Multimedia Repository has been extended to provide support for image formats that can only be read. The image library used by Vitalware allows a number of image types to be read but not written. As the format cannot be output, it is not possible to generate images of these formats, however media may be added and viewed. In particular, support for the display of DNG (Digital Negative) format is now available.
- **Scan to PDF files**: The Multimedia Repository allows documents to be scanned and saved in Vitalware. Vitalware provides support for a number of formats in which the scanned document can be saved. Support for PDF has now been added. PDF documents are now displayed in Vitalware with the first page of the document shown. For configuration options consult the Image File Extension Registry entry documentation in the Vitalware help.
- **Unique Indexing**: A new indexing Registry entry has been added that allows columns to be defined as unique. The format of the entry is:

  `System|Setting|Table|`*`table`*`|Unique Index|`*`column`*

  where *table* specifies the module and *column* denotes the column in the given table for which unique indexing is to be enabled. Once unique indexing is enabled, auto incrementation may also be specified. See the *Uniqueness and Auto-incrementation* topic in the Vitalware help for more details.
- **Lookup table permissions**: The same Lookup table may be associated with a number of fields across multiple modules, e.g. the *Yes/No* Lookup table. A new Admin Task *Set Global*

Lookup List Permissions has been added which allows the permission to be set for the Lookup table as a whole without the need of having to set it for every column.

- **POS orders received from the shopping cart**: The process to transfer orders from the shopping cart to the POS system has been enhanced to store all of the data from a shopping cart record in the Original Data of the POS record.
- **Match Lists reports**: Reference fields have been added to the Match Lists module to allow reports to be generated containing data from the associated registration modules.
- **Printing communications**: It is possible to print communications on a printer that is not the default printer. This option is available through the new Show Printer Dialogue Registry entry.
- **Manual stock number prediction**: The manual stock number prediction has been changed so that it may either increment or decrement the stock number. This is controlled by the new Stock Print Direction Registry entry.

Issues Resolved

Issue	Resolution
The global replace operational privilege, `daReplace`, is required to perform a number of functions in the Vitalware client. In particular it is required for: • Global Replace • Relocate, Revalue, Recondition, Reidentify tools • Record Merging A number of requests have been made to create a new operational privilege for record merging, as it is logically a separate activity.	A new operational privilege, `daMerge`, has been added to the set of permissions. The new permission, rather than `daReplace`, is required to perform record merging.
The **Ditto All** command allows users to replace completely the contents of an existing record. A request has been made that a new operational privilege be created since all data elements are modified. The privilege will allow System Administrators to restrict who can use the **Ditto All** command.	A new operational privilege, `daDittoAll`, has been added to the set of permissions. The new permission is required to access the **Ditto All** command.
If an image is displayed in the Multimedia tab and the user moves to the next record, where the next record contains a video format that is not supported, an error message is displayed indicating that the format is not supported. Once the error message is dismissed, the image of the previous record is still shown.	If a media format is not supported by Vitalware, the multimedia display on the Multimedia tab is now cleared.
If the multimedia display on the Multimedia tab is showing multiple thumbnails and the view attachments button is selected, all attached multimedia records are displayed in the Multimedia Repository. The selected thumbnail is not the first record displayed, forcing users to move through the multimedia records to locate the required match.	When the view attachments button is selected on the Multimedia tab, all attached records are displayed with the current record set to the thumbnail selected on the Multimedia tab.
If the Vitalware help files are invoked while the prompt language is set to French (CA), then the English help text is shown rather than the French	The French help text is now displayed when help is invoked with the prompt language set to French (CA).

Issue	Resolution
text. The French text is displayed correctly when the prompt language is set to French.	
Vitalware does not provide a way for setting a default value for an attachment field in Query mode. The Set Default Value dialogue box displays the field but does not provide an attachment button to allow records to be selected.	The Set Default Value dialogue box allows records to be attached as default values for attachment fields.
The Column Colour Registry entry does not provide a mechanism for setting the default colour for all controls in a module.	A column name of *Default* may now be used with the Column Colour Registry entry to set the default colour for all controls in a given module.
The lookup of security settings for the Security tab in property dialogue boxes may be slow when a large number of users are registered on the system.	The lookup speed for security settings has been optimised to provide faster loading of the Security tab in property dialogue boxes.
The text colour used to display records in List View where the record is read only is a light grey colour. The colour makes the text difficult to read against a white background and blue background when the record is selected.	The display of read only records in List View has been modified to use a grey background with black text. The change makes read only records more distinct visually.
The security profile generated for user `Vitalware` does not contain any record level security restrictions (by design). If the `Vitalware` account is configured to allow multiple groups, the Record Level Security settings will not reflect the group selected. Rather no restrictions will apply. This makes it difficult to test group permissions as user `Vitalware`.	The security profile generated for user `Vitalware` only provides unrestricted permissions for the first group listed, where multiple groups are supported. In general, the first group listed should be the "super-user" group (usually `Admin`).
The addition of multiple group support allows a user to be in more than one group, possibly at the same time. The Vitalware audit trail records do not contain the group of the user who updated, inserted or deleted a record. The user's group is useful for tracing why changes were allowed to a record.	The group of a user who inserts, modifies or deletes a record is now recorded in the audit trail record.
A user may set the format used to display dates via the Date Order Registry entry. The Vitalware server also has a default date display format. In general these two settings should display the date components (day,	Users may now specify a date format with the date components in any order. The order will be used for both data entry and searching.

Issue	Resolution
month, year) in the same order. If the order is different, then dates entered when searching must be in the server defined order.	
Vitalware uses the Windows Registry to try to determine the MIME type for a given file extension. If the MIME type cannot be determined, then the Vitalware Mime Registry is consulted. An issue arises where a third party application has installed a bad mime type into the Windows Registry. Multimedia records created for that application will contain an invalid mime type value.	The Vitalware Mime Registry entry has been extended to allow it to override an entry in the Windows Registry.
The log files generated by the various Vitalware server side processes may not output record based data correctly where the data is in UTF-8 format. The log files do not affect any records stored in Vitalware.	Vitalware server side processes now log UTF-8 based record data correctly.
The audit trail data generated for loading into the Audit Trail module may exceed the maximum file size when auditing has been disabled for a reasonable period.	The audit trail data is now split up into a number of files where each file is limited to the maximum file size.
If an attachments fails on a RichEdit control, then the original value is restored to the control. If the RichEdit control is then tabbed off, the RichEdit control will try an attach query again. The second lookup is not required as the RichEdit control has been restored to its original value.	When an attachments fails on a RichEdit control and the original value is restored, the second attachment lookup no longer occurs.
When the context menu is shown in List View by right clicking on a record that is not selected currently, the menu may not have the correct menu options enabled. In particular the multimedia options Save and Launch may be disabled when they should be enabled.	The Save and Launch menu options are now enabled correctly.
If an entry is selected from a ComboBox, where the associated column has a form set via the Vitalware Format Registry setting (e.g. convert to uppercase), then the format is not applied to the value selected.	The format setting is now applied for all ComboBox selections where a format has been specified.
If the value in a ComboBox is used to determine which tabs should be shown and the ComboBox is part of a hierarchy, then an error message may be displayed when a new value forces the tabs to adjust.	The error message no longer occurs when tabs are adjusted due to a change in the ComboBox value.
The Edit Resource facility does not allow Microsoft Excel documents to	The Edit Resource facility has been expanded to invoke the View

Issue	Resolution
be edited. Other Microsoft Office documents can be updated. The problem is due to Excel not registering an Edit mechanism in the Windows Registry.	mechanism for a document if an Edit mechanism is not found. The change does not guarantee the document can be modified if the View mechanism is invoked.
The Field Help dialogue box may not be displayed correctly when the option to *Save last Size* is enabled. The dialogue box shows the help window with a grey area similar in size to the help window next to it.	The correct size for the Field Help dialogue box is now maintained when *Save last Size* option is enabled.
An **Access Violation** message may be displayed when closing a crystal report that contains an image (e.g. logo) in the main report and an Vitalware multimedia image in a sub-report.	The **Access Violation** message no longer occurs when closing the report viewer.
The Field Help dialogue box may not display the correct extended information (e.g. *Column Name*, etc) the first time it is invoked. The correct information is shown for subsequent invocations.	The correct extended information is now shown the first time the Field Help dialogue box is displayed.
An **Access Violation** error message may be displayed when Copy and Paste is used to set a default value in Query mode via the Set Default Value dialogue box.	The **Access Violation** error message no longer appears when setting a default value via Copy and Paste.
If the escape key (ESC) is pressed when the Save Changes dialogue box is displayed, then the changes are saved rather than being cancelled.	The escape key now cancels changes made when the Save Changes dialogue box is displayed.
When setting default values for Query mode for CheckBox controls via the Set Default Value dialogue box, the order of the controls in the dialogue box may not be the same as on the tabs.	The order of CheckBox controls in the Set Default Value dialogue box is now the same as on the Query tabs.
When inserting a new record from Query mode, the Status Bar at the bottom of the module form may indicate a number of records are selected. No records should be selected since the insertion started from Query mode.	The Status Bar no longer indicates any records are selected when an insertion is started from Query mode.
If List View is displaying more columns than can be shown on the screen and the screen is scrolled to the right and a row number is selected, then the List View scrolls to the left most column. The scrolling of the columns makes it difficult to compare data in the right most columns as they have scrolled off the screen.	List View no longer scrolls to the left most column when a row is selected.
When a new group is created for a set of record, the group name is not	The Group Name for newly created groups is now added to the

Issue	Resolution
added to the Lookup List of group names until the next day.	Lookup List when the group is saved.
The Reports Properties dialogue box may display an incorrect *Modified* time for the report file when daylight savings is in effect. The time displayed is one hour behind the correct time.	The correct *Modified* time for the report file is now displayed.
The error message **Cannot load image resource "ListImg"** may be displayed when viewing default values via the Set Default Value dialogue box. The error only occurs when an invalid value is set on a date field.	The error message no longer appears when an incorrect default value is set on a date field.
If the **Edit>Clear** command is invoked in a grid in Display mode, then the value with focus is cleared in the grid, however Vitalware does not change to Edit mode and if the record is refreshed the cleared value is re-displayed.	When the **Edit>Clear** command is invoked in a grid, Vitalware changes to Edit mode and the value with focus is cleared.
The error message **TexAPI Error. Cannot allocate memory (Number 100) at offset 0** may occur when the **Viewing Attached>Selected Records** command is invoked on a large number of selected records (e.g. 150,000).	The error message no longer appears when the **Viewing Attached>Selected Records** command is invoked on a large number of selected records.
An **Access Violation** message may be displayed when a new version of ImageMagick is installed on a user's computer and Vitalware is configured to use the new version. The message is not displayed if the version of ImageMagick distributed with Vitalware is used.	The error message is no longer displayed if a newer version of ImageMagick is used.
If a language other than All Languages is selected in the Report Properties dialogue box for a multi-lingual system (e.g. English/French), then the Language selection is not saved correctly when the properties are exited.	The Language selected in the Report Properties dialogue box for a multi-lingual system is now saved correctly.
The **Vitalwaresecurity** server side program may not be run automatically when Security Registry entries are removed. The issue only occurs when a Registry entry is changed from a security based entry to another type of Registry entry.	**Vitalwaresecurity** is now run automatically when Security Registry entries are changed to another type of Registry entry.
When an image is added to the Multimedia Repository, the modified time on the original image may be changed. The original image itself is not changed, just the modified time.	The modified time is no longer changed on the original image when added to the Multimedia Repository.

Issue	Resolution
If a macron character is entered into a RichEdit control, then all text stored after the macron character will be displayed in a different font. The issue only occurs when the text is viewed in Display mode.	Text after a macron character is now displayed in the correct font.
The list of fields available for reporting may contain some columns that cannot be viewed in Details View. Since the column cannot be displayed, values cannot be entered so the columns should not be selectable for reporting.	Only columns that are displayed can be selected for reporting.
Under certain circumstances it is possible to save a record where a hierarchy of values does not exist in the Lookup List module, without being asked to confirm the new combination. The issue only arises if the new combination matches an existing entry except the new combination has some empty values.	The user is now prompted to confirm the new hierarchy of values.
An **Access Violation** error message may be displayed when switching to Page View if the XSLT report used to generate the view does not exist on the Vitalware server.	A suitable error message is now displayed if the XSLT report does not exist on the Vitalware server.
The Vitalware Import facility does not honour the Column Access Modifier Registry entry settings. Hence users may not be able to update values for which they have been granted dynamic access.	The Import facility now honours the Column Access Modifier Registry entry.
A **Grid index out of range** error message may be displayed when the Page Down key is pressed to try and move past the last page of records.	The error message no longer appears when trying to "Page Down" past the last page.
The keyboard shortcut to insert the current date (`CTRL+;`) does not work on a French keyboard.	The keyboard shortcut for the current date now works correctly for French keyboards.
When switching prompt languages from English to French, dates displayed with the months as text did not change to use the text of the new prompt language. The issue only occurs if the language is changed in the current session.	The text values for months in dates are now changed correctly when the prompt language is changed.
If a Lookup List is displayed on a grid control which in turn is controlled via another grid control (a so called Nested Form construct), then the list may not be restricted to the set of values valid for the existing hierarchy.	The correct list of values is now displayed for hierarchies represented by two grids linked together.

Issue	Resolution
If the Down arrow key is pressed in an empty ComboBox control, the list of values may not scroll through the control. The issue only arises if a value is not selected in the ComboBox control.	The Down arrow key now scrolls though the list of values in a ComboBox control when a value is not selected initially.
When a scheduled export ran that produced no output, there was nothing to indicate that it had run.	A new parameter (`-z`) has been added to `vwexport` to indicate that an empty export should produce a file indicating that no records matched.
When using postal code lookups to import address information, the keying of the street number for the street address would trigger a second lookup to verify that the postal code was still valid.	The functionality has been changed so that the default action is not to trigger a second verification. This action can be altered by setting the new Ignore Address Change Registry entry.
When a scheduled export ran, there was no notification of it completing.	A new email field has been added to the schedule and if filled, an email is sent to the address indicating the results of the export.
On rare occasions the status of an order would not be set to `Complete` when a certificate was printed.	The status of the order is now set to `Complete`
On rare occasion the discount would not calculate until the user exited the Products group box.	The discount is calculated when the user exits the *Discount* field.
On rare occasions a credit card purchase was not refunded when a POS record was cancelled.	The credit card is now refunded.
The refund type could be set on a POS record even though no refund was issued.	The refund type is only set when a refund is issued.
On rare occasions a user could be asked to insert a value into a Lookup List even though the value already existed in the list.	The user is no longer asked if they wish to insert values that already exist in the Lookup List.
When a row was copied and pasted into the POS module *Sales* grid, it was possible that the End of Day amount for that day would not balance.	End of Day now balances when rows are copied and pasted into the *Sales* grid.
An invoiced POS transaction with a reversed order could contain an incorrect invoice total.	The invoice total is correct for Transactions with revered orders.
If two users accessed the same POS transaction at the same time, one to print a receipt and the other to print certificates, it was possible for a duplicate POS record to be created.	One user will now get a popup indicating that another user is editing the record.

Issue	Resolution
A maintenance / amendment would be allowed to be applied even if the POS record it was associated with could not be updated.	The maintenance / amendment is now rolled back if the POS record is not able to be updated.
On occasions the overnight certificate printing scripts would not print the expected number of certificates.	The expected number of certificates is printed.
On occasions End of Day would not balance correctly if a partial invoice payment was made.	End of Day now balances correctly when partial invoice payments are made.
The entry of a negative payment amount was accepted.	Negative payment amounts are not accepted.
On occasions the status for a Refund Item would not get set to `Complete` even though the refund had been issued.	The status of the Refund Item is set to `Complete` when the refund is issued.
On rare occasions when printing a certificate, duplicate Ledger and Order records could be created causing End of Day to not balance.	Duplicate Order and Ledger records are no longer created.
The entry of a discount greater than a products cost was allowed.	Only a discount up to the full product cost is allowed.
When making a second cash payment on a transaction that had a previous cash payment which required change, the incorrect change amount could be displayed.	The correct change amount is displayed.
When viewing Historic records, the *Stock Issued For Current Record* may not display the stock issued for the Historic record correctly.	The stock issued for the Historic record now displays correctly.
When running a scheduled export via cron, the start date or the end date could be incorrectly set on the export record.	The start date and the end date are correctly set.
On occasions the *Date Of Event* field in POS would not display the entered date in the selected output format.	The *Date Of Event* displays in the correct output date format.
When logging in it was possible to select the OK button more that once, resulting in multiple licences being used.	The OK button may only be selected once.
When returning a range (subset) of issued stock that had previously been issued and returned, a message was displayed indicating that the parent stock record could not be found.	The returned stock is now accepted and no message is displayed.
When keying stock records it was possible to enter an invalid stock number.	Invalid stock numbers are no longer accepted.

Issue	Resolution
After swapping printers for certificates or receipts, the certificates and receipts still printed to the previous print location.	The certificates and receipts print to the new print location.
It was possible for a product to be issued prior to all of the mandatory data being supplied.	Products are not issued until all mandatory fields have been completed.
When using the show validation errors field colouring, the colouring was not reset when entering Insert mode.	Fields colours are reset when entering Insert mode.
Changes to page size and orientation in the Communications module were not reflected in the next print of a letter.	The page size and orientation are correctly reset for the next print of a letter.
When module field help is turned on and module caching is being used, subsequent module displays increase the size of the module.	The module remains the same size.

Upgrade Notes

The upgrade from Vitalware Version 2.2.02 to Vitalware 2.3.01 involves a number of steps. Please follow the instructions below carefully.

Do not skip any steps under any circumstances.

Before proceeding with the update please ensure that a complete backup of the Vitalware server exists and is restorable.

There are four components that require upgrading:

- Texpress (the database engine)
- TexAPI (web services)
- Vitalware Server (the application)
- Vitalware Client (the client)

The notes below detail how to upgrade all systems. Check the Releases table for Client specific notes.

In the notes below, *clientname* refers to the name of the client directory for the current installation. The term `~vw` is used to refer to user `vw`'s home directory. This is normally `/home/vw`.

Stopping Vitalware services

1. Log in as `vw`
2. Enter `client` *`clientname`*
3. Enter `ls -l loads/*/data* local/loads/*/data*`
4. Check that each `data` directory is empty and that no `data.t` files exist.
   If `data.t` files do exist, please wait for the loads to drain before proceeding.
5. Enter `vwload stop`
6. Enter `vwweb stop`
7. Enter `texlicstatus`
   Make sure no one is using the system.
   The upgrade will not complete successfully if users are accessing data.

Record Session

Each step in the upgrade process produces detailed output. In most cases this output will exceed the size of the screen. It is `strongly` recommended that the output of the upgrade session is recorded, so if errors occur, the output can be examined.

1. Enter `script /tmp/output-2-3-01`

A new shell will start and all output recorded until the shell is terminated.

Installing Texpress

Installing Texpress 8.3 is only required for the first client upgraded to Vitalware 2.3.01. Once Texpress 8.3 has been installed, this section may be skipped for subsequent upgrades.

1. Enter `cd ~vw`
2. Enter `mkdir -p texpress/8.3.xxx/install`
3. Enter `cd texpress/8.3.xxx/install`
4. Obtain the appropriate Texpress version for your Unix machine.
   Save the release in `~vw/texpress/8.3.xxx/install`, calling it *texpress.sh*.
5. Enter `sh texpress.sh`
   The Texpress release will be extracted.
6. Enter `. ./.profile`
7. Enter `bin/texinstall~vw/texpress/8.3.xxx`
   The Texpress installation script will commence.
8. Enter `cd ~vw/texpress/8.3.xxx`
9. Enter `. ./.profile`
10. Enter `bin/texlicinfo`
    Obtain your Texpress licence code and place it in a file called `.licence`.
11. Enter `bin/texlicset< .licence` to install the licence.
12. Enter `\rm -fr install`
13. Enter `cd ~vw/texpress`
14. Enter `ln -s 8.3.xxx 8.3`

Upgrading KE TexAPI

Installing TexAPI is only required for the first client upgraded to Vitalware 2.3.01. Once TexAPI has been installed, this section may be skipped for subsequent upgrades.

1. Enter `cd ~vw/texpress`
2. Enter `mkdir 6.0.xxx`
3. Obtain the appropriate TexAPI version for your Unix machine.
   Save the release in `~vw/texpress`, calling it *texapi.sh*.
4. Enter `sh texapi.sh -i~vw/texpress/6.0.xxx` (expand the *~vw*).
5. Enter `\rm -f texapi`
6. Enter `ln -s 6.0.xxx texapi`
7. Enter `\rm -f texapi.sh`

Upgrading Vitalware Server

1. Enter `cd ~vw/clientname`
2. Enter `mkdir install`
3. Enter `cd install`
4. Obtain the appropriate Vitalware server version bundle.
   Save the release bundle file in `~vw/clientname/install` calling it *vw.sh*.
5. Enter `sh vw.sh`
   The Vitalware release will be extracted.
6. Enter `. ./.profile`

7. Enter `bin/vwinstall`*`clientname`*
   The Vitalware installation script will commence.
8. Enter `cd ~vw/`*`clientname`*
9. Enter `cp .profile.parent ../.profile`
10. Enter `. ../.profile`
11. Enter `client `*`clientname`*
12. Enter `vwreindex`
13. Removal of the temporary directory (and its contents) is recommended:
    Enter `\rm -fr install`
14. Enter `upgrade-2-3-01`
    The client will now be upgraded to Vitalware 2.3.01. If you are upgrading from a version prior to Vitalware 2.2.02, you must run the upgrade scripts for all versions after the old version `before` running the Vitalware 2.3.01 upgrade.

Starting Vitalware services

1. Enter `vwload start`
2. Enter `vwweb start`

Record Session

The recording of the upgrade session may now be terminated.

1. Enter `exit`

The session output is available in `/tmp/output-2-3.01`.

Upgrading Vitalware Client

Vitalware 2.3.01 does not require the new Windows client to be installed on every machine for network installations. Updating the network server is sufficient. For standalone installations a new client is required on each machine. To upgrade the Vitalware Client follow the Installing Vitalware Client notes.

Vitalware®
Vital Records Management

The World's Premier Vital Records Software

Vitalware Documentation

Supplementary Media

Document Version 1.1

Vitalware Version 2.2.02

KE Software

www.kesoftware.com

Contents

SECTION 1

Overview

The Vitalware Multimedia repository stores and manages a wide range of document types. These may be:

- In electronic format (e.g. images, Word documents, video, etc.).
- URLs identifying a resource on the World Wide Web.
- An identifier used to locate a particular resource (e.g. the ISBN for a book, or the location of a slide in the slide library).

Each record in the Multimedia repository describes exactly one resource. Data describing the resource may be added to the record, allowing quite detailed information to be associated with it (as a minimum, the full set of Dublin Core fields is available).

An issue arises when there's a need to store resources *associated* with a Multimedia record. For example, let's say a Multimedia resource is an image of a page containing text. Along with the image it may be desirable to have a text document that contains the words in the image so that users may view the image of the text but also have the option of reading the text in the associated document.

Prior to Vitalware 2.2.02, there was only one way to implement such a solution:

1. Each associated resource required its own Multimedia record.
2. A relationship was then established between the master resource and the associated resources via either a text field in the Multimedia record or via an over-arching record in another module that links the resources.

In some cases the setting up of such structures is overkill. In the example above, the text document containing the words in the image may not be important enough to warrant its own Multimedia record.

Other examples:

- The master resource is an image and a cropped thumbnail is required to display on the web.

  The Multimedia repository generates images of various resolutions when an image is added but each of these resolutions is an exact copy of the master. In some instances it may be desirable to have cropped versions of the master, or even a different image completely.
- The master resource is a video. There may also be a separate audio track describing the video and an image of the first frame of the video.

  Rather than creating separate Multimedia records for the audio and image resources it may be more appropriate to store them with the master video itself.

The purpose of supplementary media is to allow associated resources to be stored with a master resource in the same Multimedia record. Supplementary media does not replace the use of over-arching records to related multimedia where each resource is important in its own right, rather it provides a mechanism for storing other media with the master resource that may be used along with the master resource.

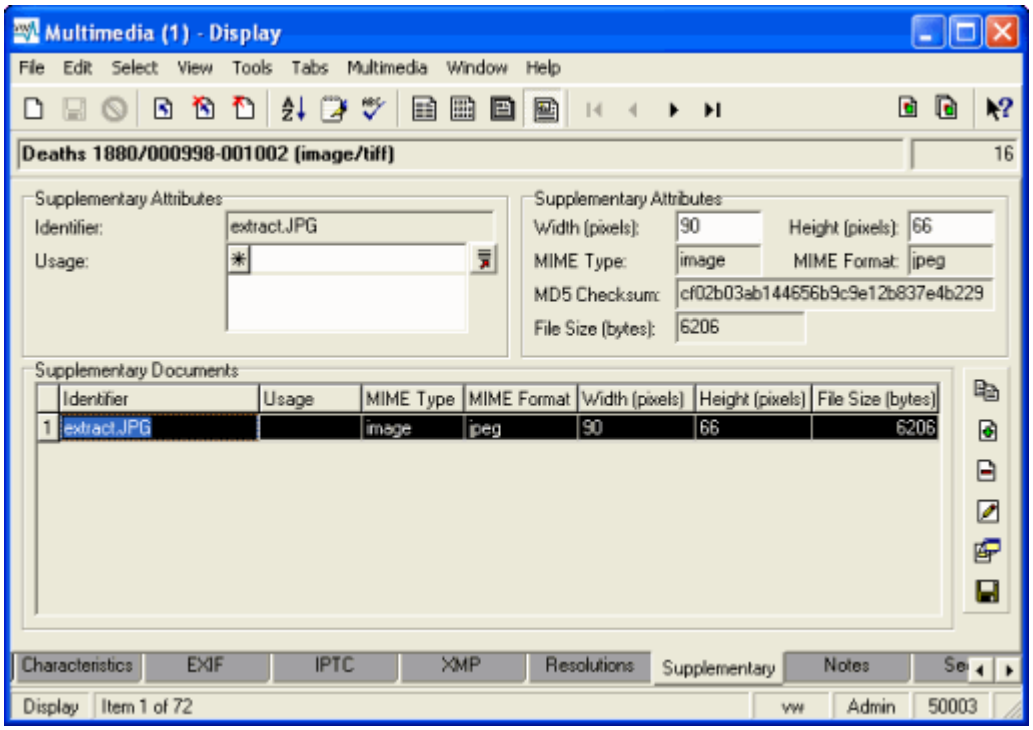
In order to provide support for supplementary media, a Supplementary tab has been added to the Multimedia module. The tab is similar to the Resolutions tab and allows media to be added, deleted, modified, viewed and saved. The Multimedia drop-down menu available in each module has been extended to allow supplementary media to be viewed and saved. It is also possible:

- To import supplementary media using the Vitalware Import facility.
- To use supplementary media in reports.

SECTION 2

Supplementary tab

All management of supplementary media is performed via the Supplementary tab in the Multimedia module. The bottom half of the tab consists of a *Supplementary* table that lists all supplementary media. The set of fields above the table display data about the media currently selected in the *Supplementary* table:



The following fields are available:

Field Name	Description
Identifier	The file name given to the media on the Vitalware server. This must be provided and must be unique in the *Supplementary* table for the current record. When adding new media the name of the file being imported is used as the default identifier. An identifier may consist of any characters, including spaces, except for: `\ / : * ? " < > |`
Width (pixels)	If the supplementary media is an image, the width in pixels is calculated automatically and stored. For other types of media that have a width in pixels (e.g. video) the value may be entered manually.
Height (pixels)	If the supplementary media is an image, the height in pixels is calculated automatically and stored. For other types of media that have a height in pixels (e.g. video) the value may be entered manually.
MIME Type	The type of the supplementary media. Vitalware calculates the MIME Type automatically and adds the value to this field. The MIME Type is a high level term describing the overall category into which the media belongs. Example values are: image, video, audio, application, etc. MIME Types available in Vitalware are defined by RFC 2046.
MIME Format	The format used to store the supplementary media. The value is used to determine how the media should be decoded for viewing, playing, etc. For each MIME Type there is a wide range of available formats. As with the MIME Type, Vitalware calculates the MIME Format automatically and adds the value to this field.
MD5 Checksum	A calculated value used to determine whether the media has been modified. Vitalware calculates the checksum automatically and adds it to this field. The checksum should be used where the authenticity of the supplementary media needs to be verified.
File Size (bytes)	The size of the supplementary file in bytes. Vitalware calculates the size of the media automatically and adds the value to this field. The value may be used to determine download times, storage requirements, etc.
Usage	A list of values outlining what the supplementary media may be used for. The usage field allows supplementary media to be searched for based on its purpose. A Look-up List is provided to allow some vocabulary control over the available values.
Notes	An area where notes about the supplementary media may be stored. The notes may describe what the media contains, or it may be information that is displayed along with the media when displayed on a website.

Only the *Usage*, *Notes*, *Width* and *Height* fields may be altered. All other fields are calculated by Vitalware automatically and cannot be modified.

SECTION 3

Supplementary media functionality

The Supplementary tab provides a rich set of functions allowing media to be:

Permissions

In order to manipulate supplementary data, users require permission to alter the *Supplementary_tab* column. They also require certain multimedia operations depending on the command to be performed and must be able to modify the Multimedia record with appropriate record level permissions. The table below outlines the permissions required for each of the supplementary functions available:

Function	Column Permissions	Record Permissions	Multimedia Permissions
Add	`dvInsert, duInsert, dvEdit, duEdit`	`Edit`	`Add`
Import	`dvInsert, duInsert, dvEdit, duEdit`	`Edit`	`Add`
Delete	`dvEdit, duEdit`	`Edit`	`Delete`
Edit	`dvEdit, duEdit`	`Edit`	`Add`
View	`dvDisplay`	`Display`	
Save	`dvDisplay`	`Display`	
Update	`dvEdit, duEdit`	`Edit`	`Update`

Note:

- Column Permissions are controlled via the `Column Access` Registry entry.
- Record Permissions are based on record level security settings.
- The `Security` Registry entry may be used to set the default permissions.
- The record level permissions on an individual record may be altered by users with sufficient privileges.
- Multimedia permissions are determined by the `Multimedia|Operations` Registry entry.

See the Vitalware Help for details.

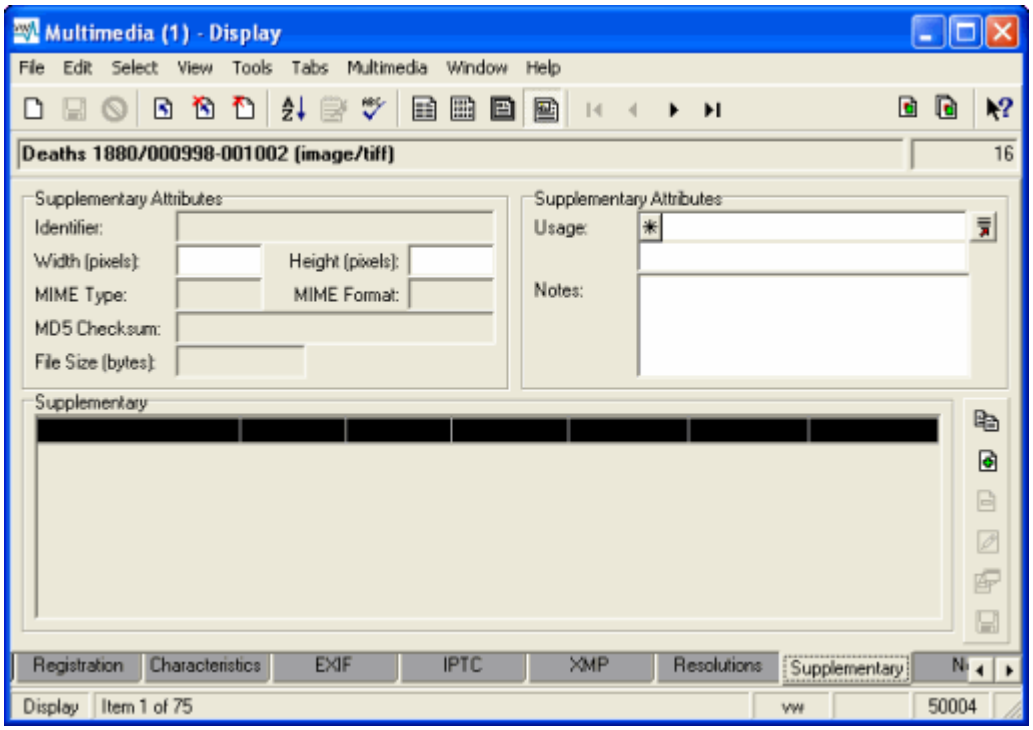
How to add supplementary media

One way to add supplementary media to a record is to take a copy of the master resource and modify it in an editor. When the modified image is saved, Vitalware asks whether it should be added as supplementary media. An affirmative response allows an identifier to be specified, after which the media is appended to the *Supplementary* table.

In order to add supplementary media in this way the Multimedia record must have a master resource available in electronic form.

In the Multimedia module:

1. Locate the record to which supplementary media is to be added.
2. Select the **Supplementary** tab:



3. Select **Multimedia>Add>Supplementary** in the Menu bar

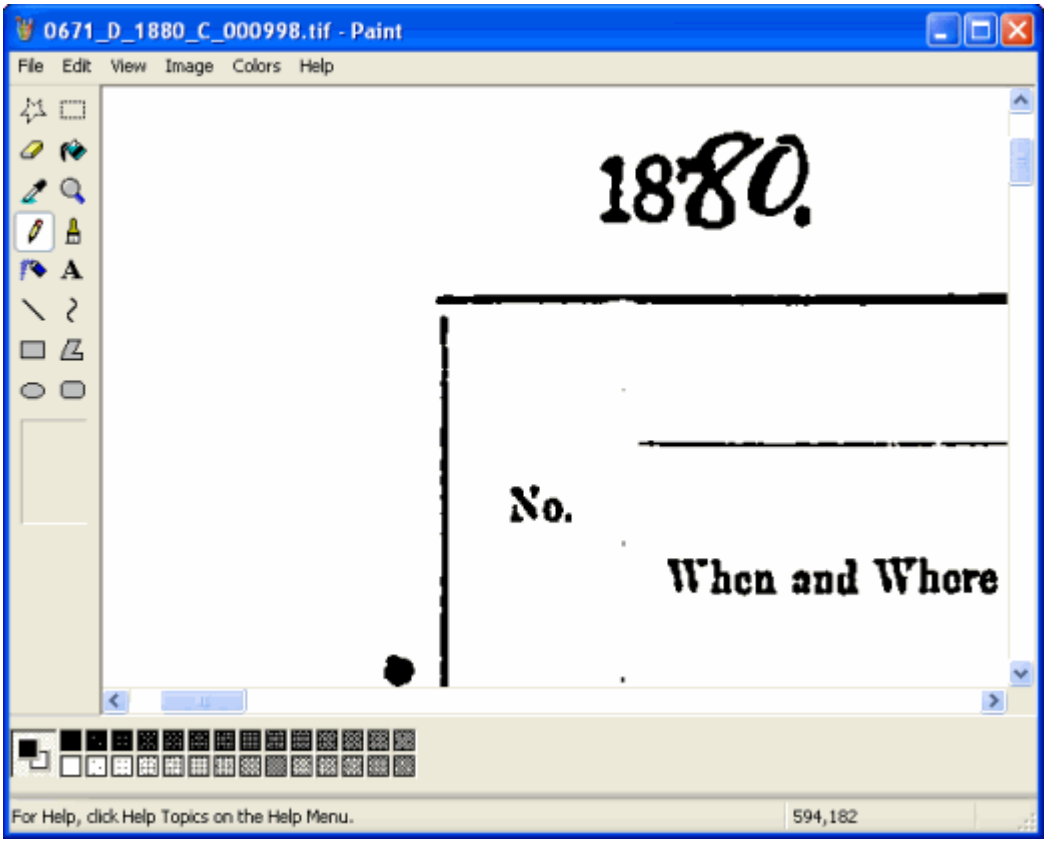
   -OR-

   Select **Add Supplementary**  from the Toolbar beside the *Supplementary* table

   -OR-

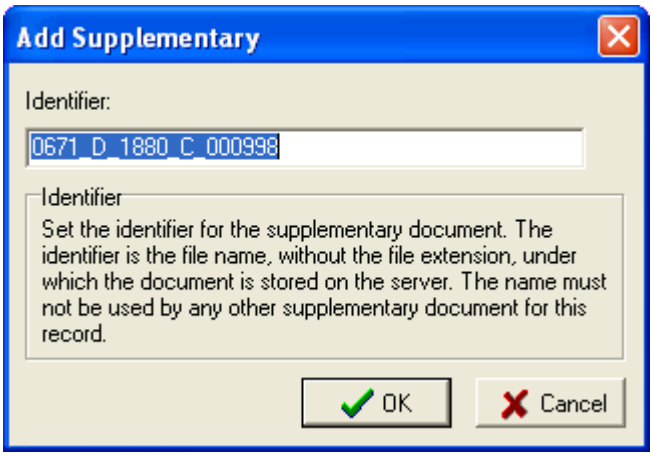
   Use the keyboard shortcut, `ALT+M+A+S`.

   The master resource displays inside the editor associated with the resource type (in this example, the Paint application is associated with jpg images):



4. Modify the resource and save the image.
5. Either exit the editor or switch back to Vitalware.

   The Add Supplementary dialogue box displays:



6. Accept or edit the *Identifier* to use for the supplementary resource.

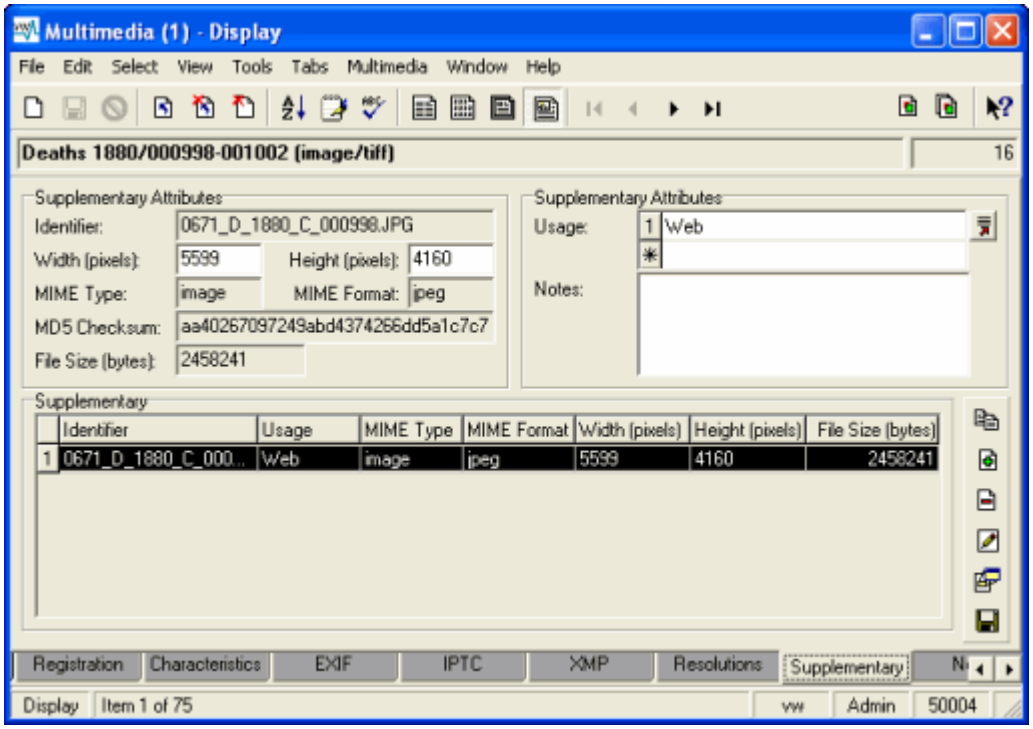
   The *Identifier* is the file name under which the supplementary media is stored on the Vitalware server.

7. Select 

.

   The new media is appended to the *Supplementary* table.

8. Add any *Usage* and *Notes* data and save the record:

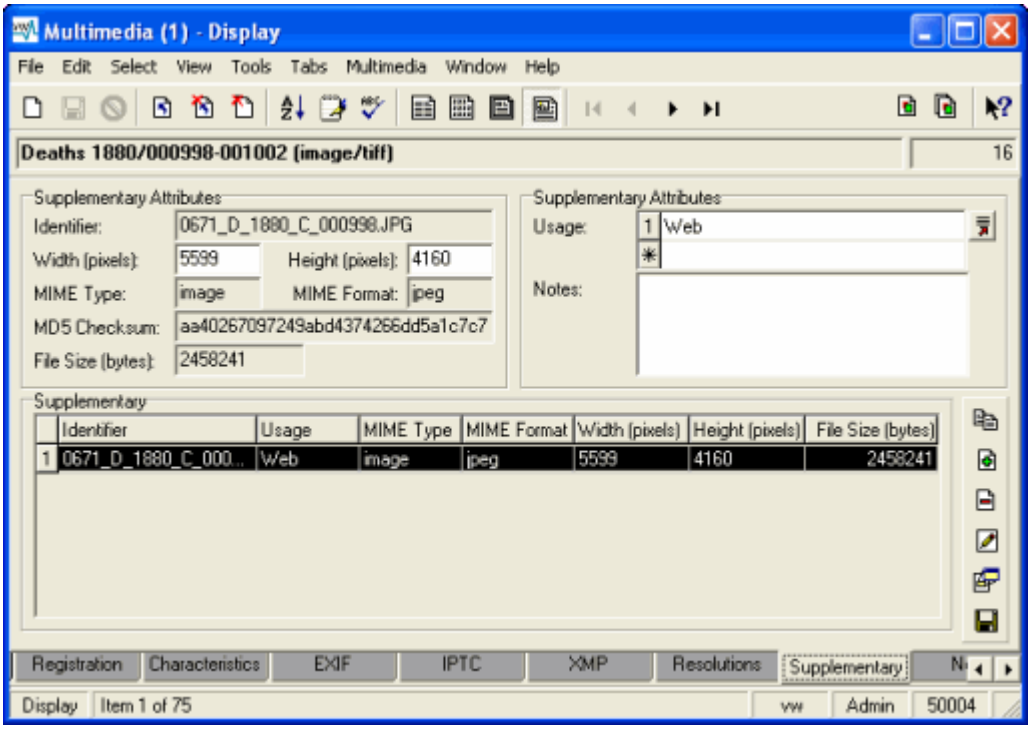

Multimedia (1) - Display
File Edit Select View Tools Tabs Multimedia Window Help
Deaths 1880/000998-001002 (image/tiff)
16
Supplementary Attributes
Identifier: 0671_D_1880_C_000998.JPG
Width (pixels): 5599
Height (pixels): 4160
MIME Type: image
MIME Format: jpeg
MD5 Checksum: aa40267097249abd4374266dd5a1c7c7
File Size (bytes): 2458241
Supplementary Attributes
Usage: 1 Web
Notes:
Supplementary
Identifier
Usage
MIME Type
MIME Format
Width (pixels)
Height (pixels)
File Size (bytes)
1 0671_D_1880_C_000... Web image jpeg 5599 4160 2458241
Registration
Characteristics
EXIF
IPTC
XMP
Resolutions
Supplementary
Display
Item 1 of 75
vw
Admin
50004

How to import supplementary media

If the supplementary media already exists as an electronic file outside of Vitalware, it can be imported into the *Supplementary* table. The process is very similar to adding a master resource to a Multimedia record.

In the Multimedia module:

1. Locate the record into which supplementary media is to be imported.
2. Select the **Supplementary** tab:



3. Select **Multimedia>Add>File** in the Menu bar

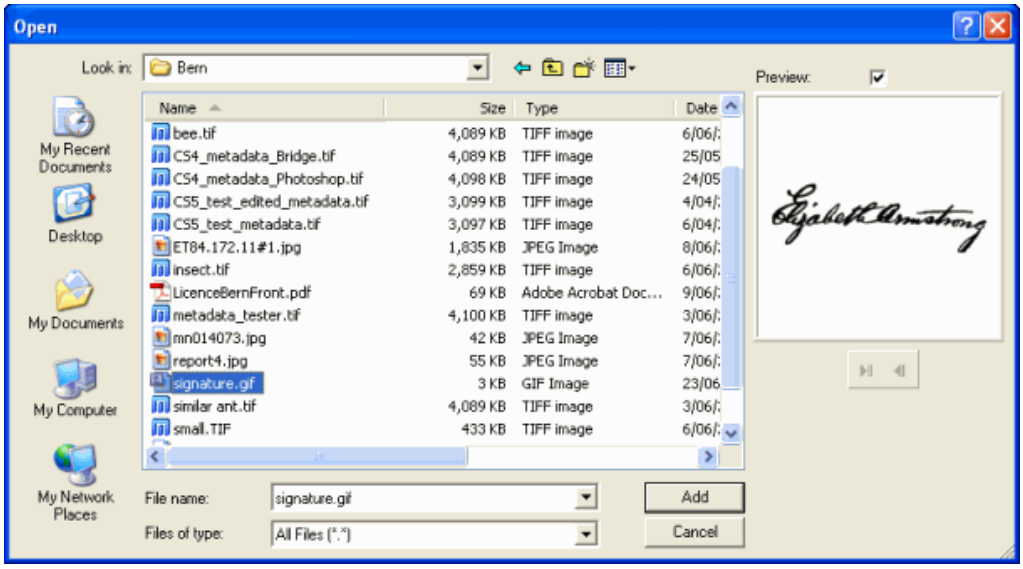
   -OR-

   Select **Import Supplementary**  from the Toolbar beside the *Supplementary* table

   -OR-

   Use the keyboard shortcut, `ALT+M+A+F`.

   The Open dialogue box displays.
4. Locate and select the media file to be imported:



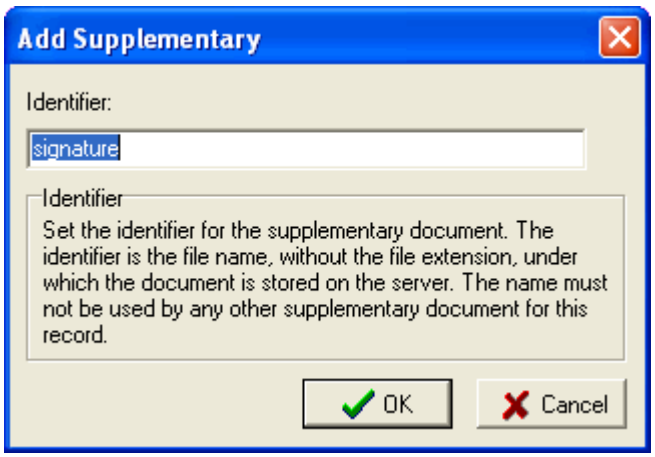
5. Click 

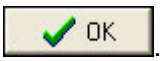
.

   The Add Supplementary dialogue box displays.

6. Enter the *Identifier* to use for the resource.

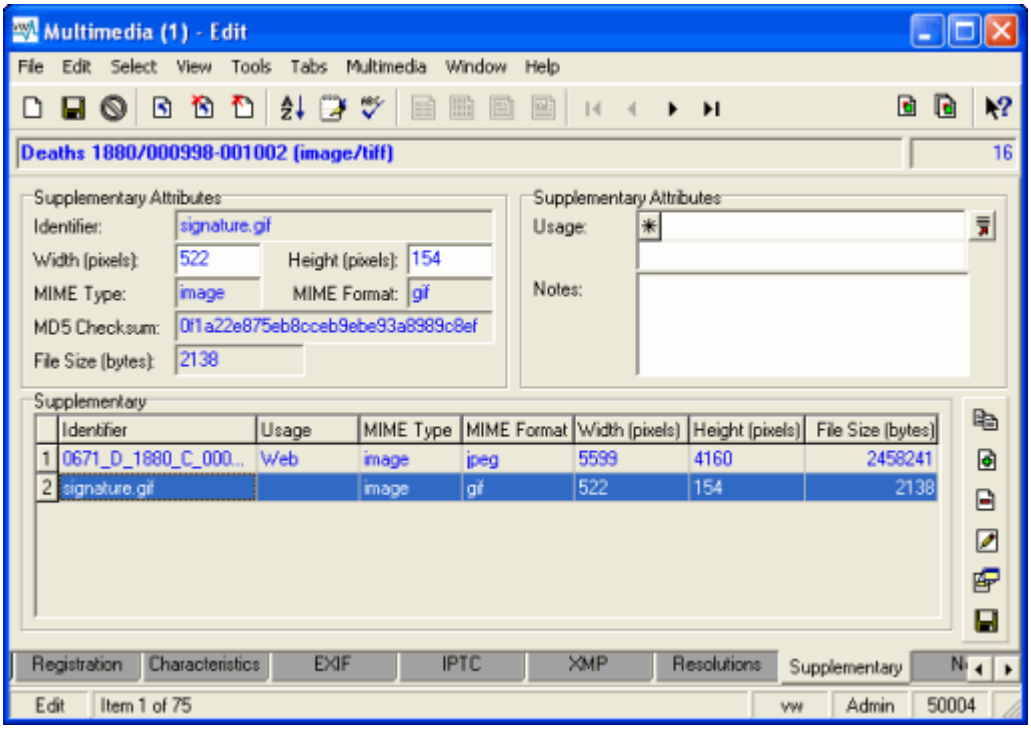
   The *Identifier* is the file name under which the supplementary media is stored on the Vitalware server:



7. Select 

.

   The file is imported and appended to the *Supplementary* table:



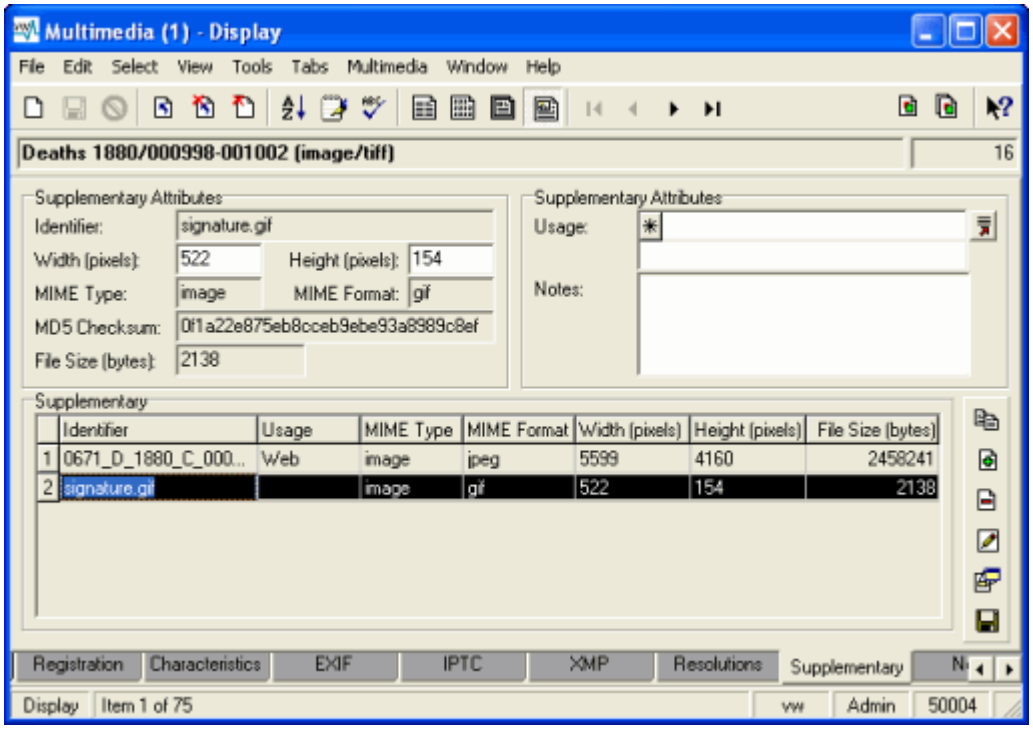
8. Add any *Usage* and *Notes* data and save the record.

How to delete supplementary media

Supplementary media may be removed from the *Supplementary* table. When the Multimedia record is saved, any media removed will be deleted from the Vitalware server permanently.

In the Multimedia module:

1. Locate the record with the supplementary media to be deleted.
2. On the Supplementary tab, click the row in the *Supplementary* table to be deleted:



3. Select **Multimedia>Delete Resource** in the Menu bar

   -OR-

   Select **Delete Supplementary**  from the Toolbar beside the *Supplementary* table

   -OR-

   Use the keyboard shortcut, `ALT+M+D`.

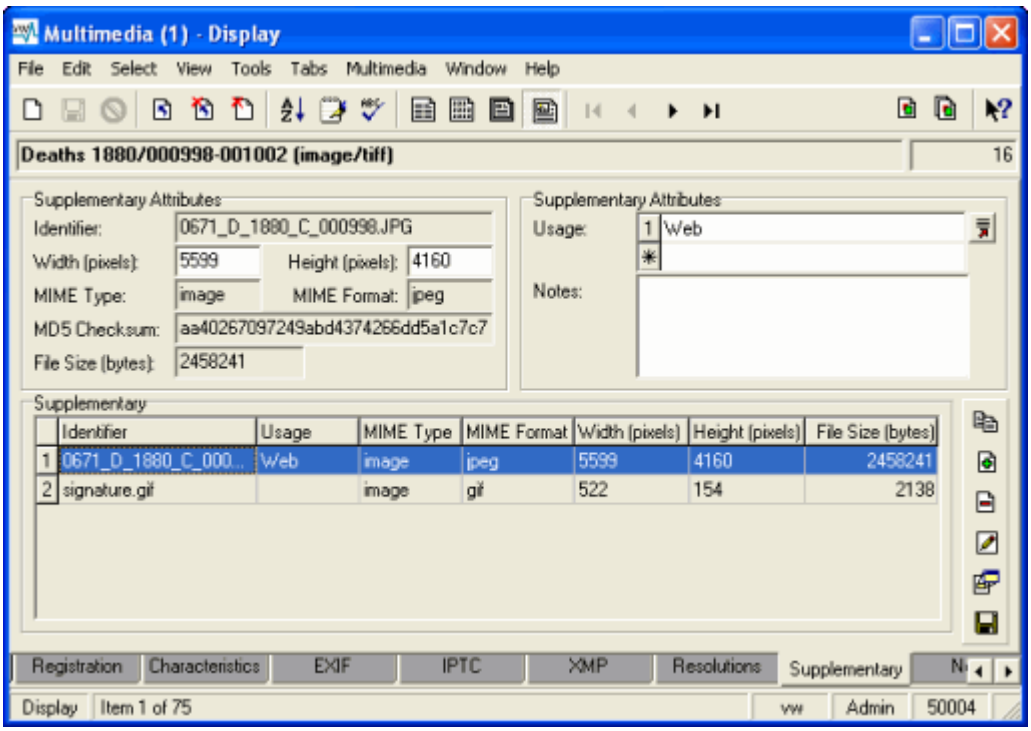
   The selected media is removed from the *Supplementary* table.
4. Save the record.

   The media is removed from the Vitalware server.

How to edit supplementary media

In the Multimedia module:

1. Locate the record with the supplementary media to be edited.
2. On the Supplementary tab, click the row in the *Supplementary* table for the media to be edited:



3. Select **Multimedia>Edit Resource** in the Menu bar

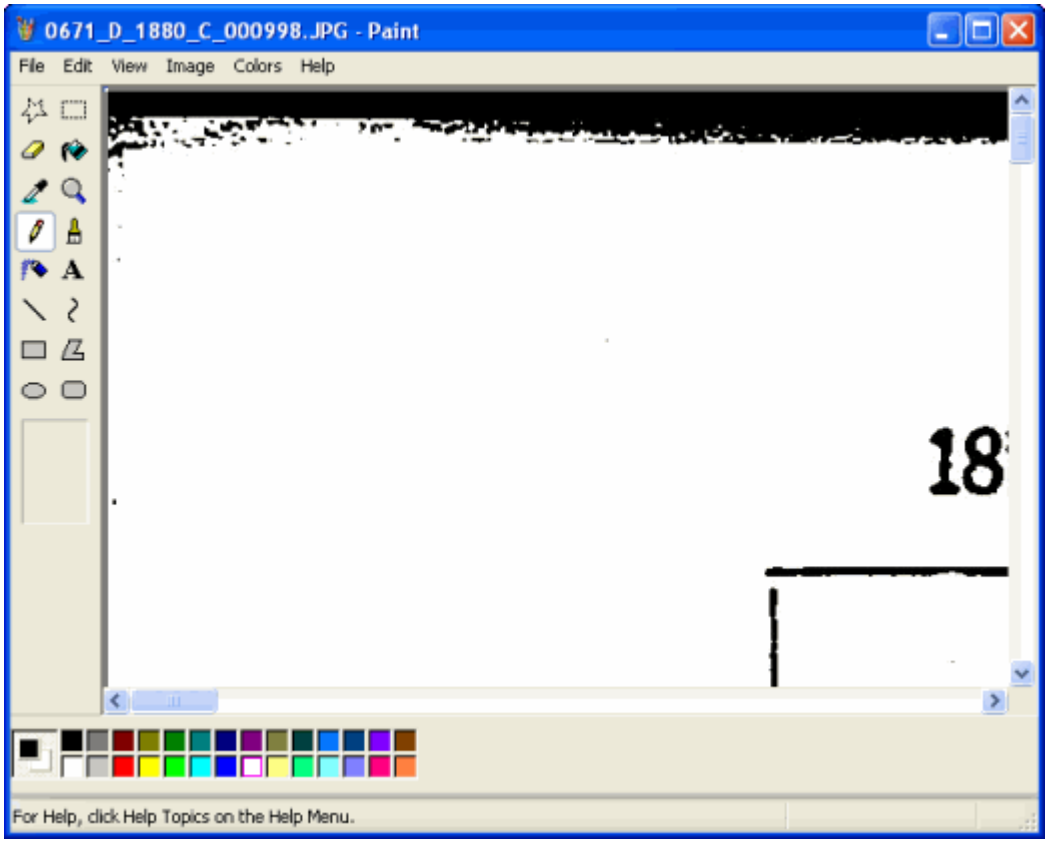
   -OR-

   Select **Edit Supplementary**  from the Toolbar beside the *Supplementary* table

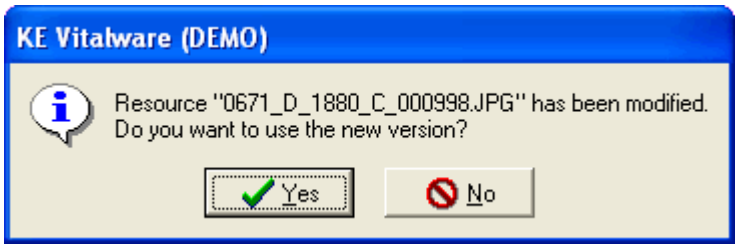
   -OR-

   Use the keyboard shortcut, `ALT+M+E`.

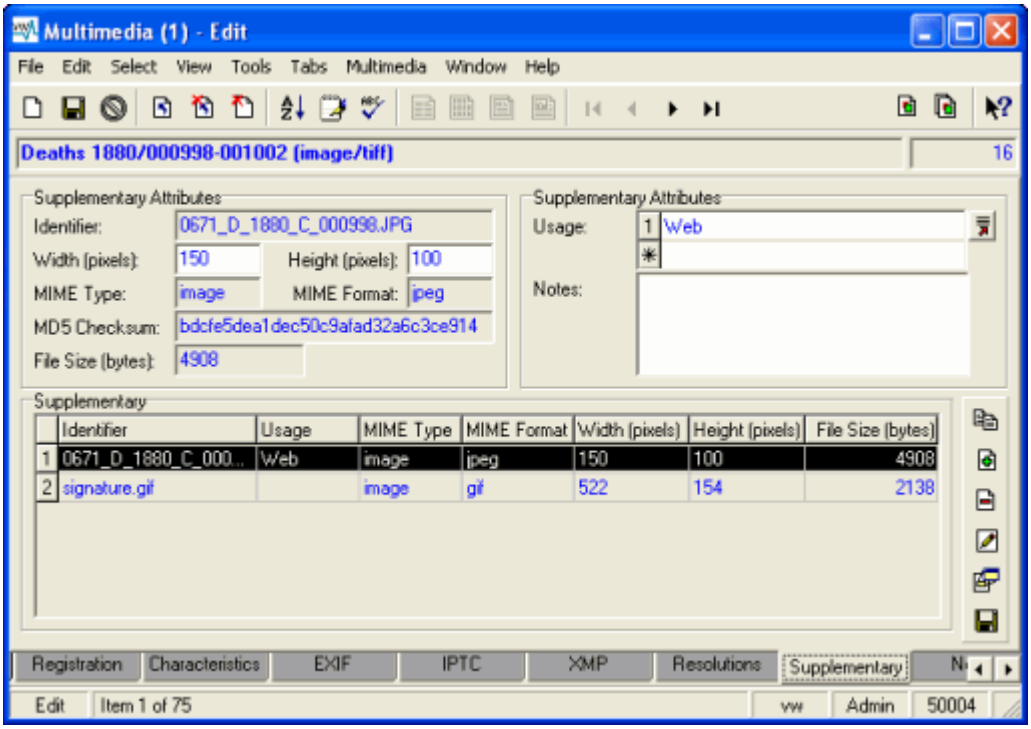
   The selected supplementary resource displays inside the editor associated with the resource type (in this example, the Paint application is associated with jpg images):



4. Modify the resource and save the image.
5. Close the editor and switch back to Vitalware.
   A dialogue box requesting confirmation of the changes displays:



6. Select  .
   The media details are updated in the *Supplementary* table:



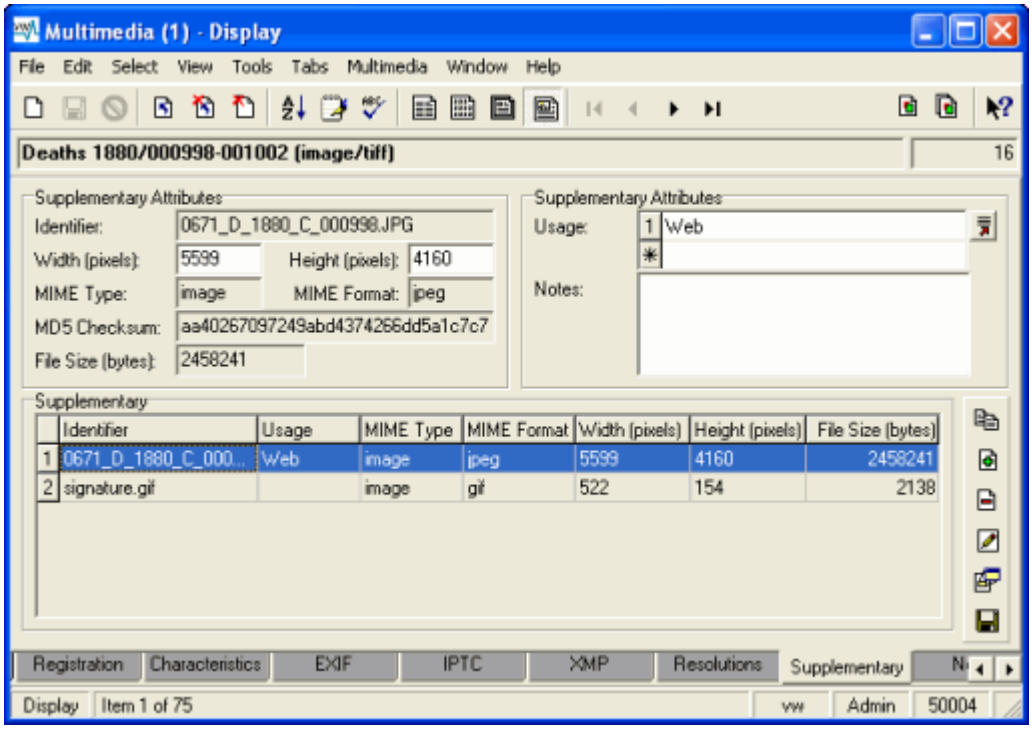
7. Save the record.
   The modified media is saved to the Vitalware server.

How to view supplementary media

Supplementary media can be viewed using external applications.

In the Multimedia module:

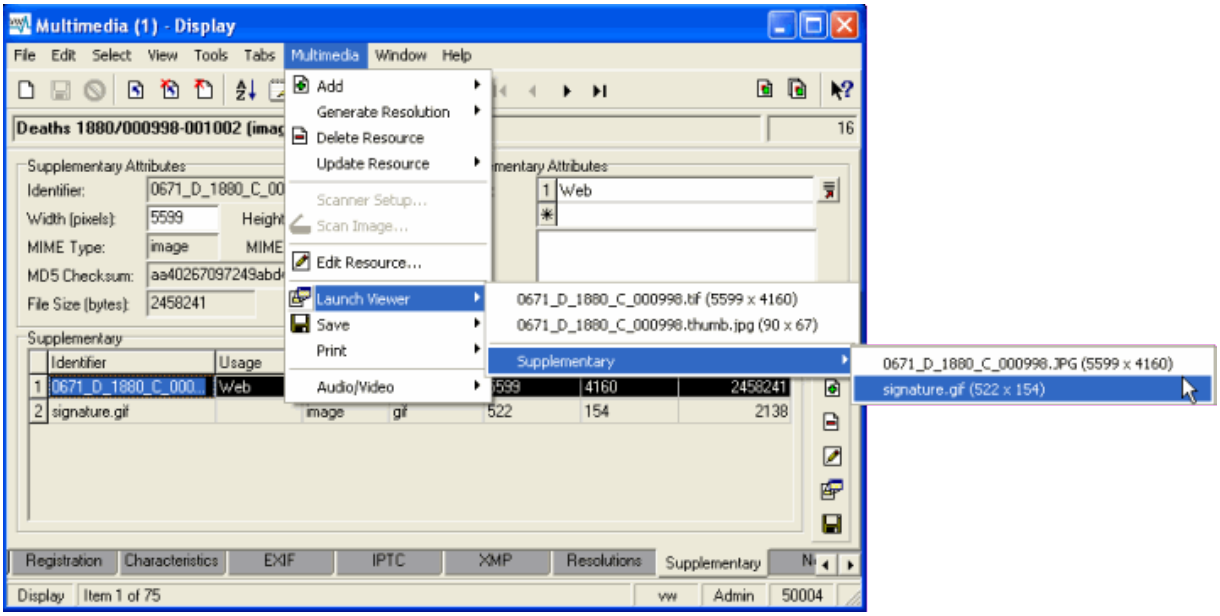
1. Locate the record with supplementary media to be viewed.
2. On the Supplementary tab, click the row in the *Supplementary* table with the media to be viewed:



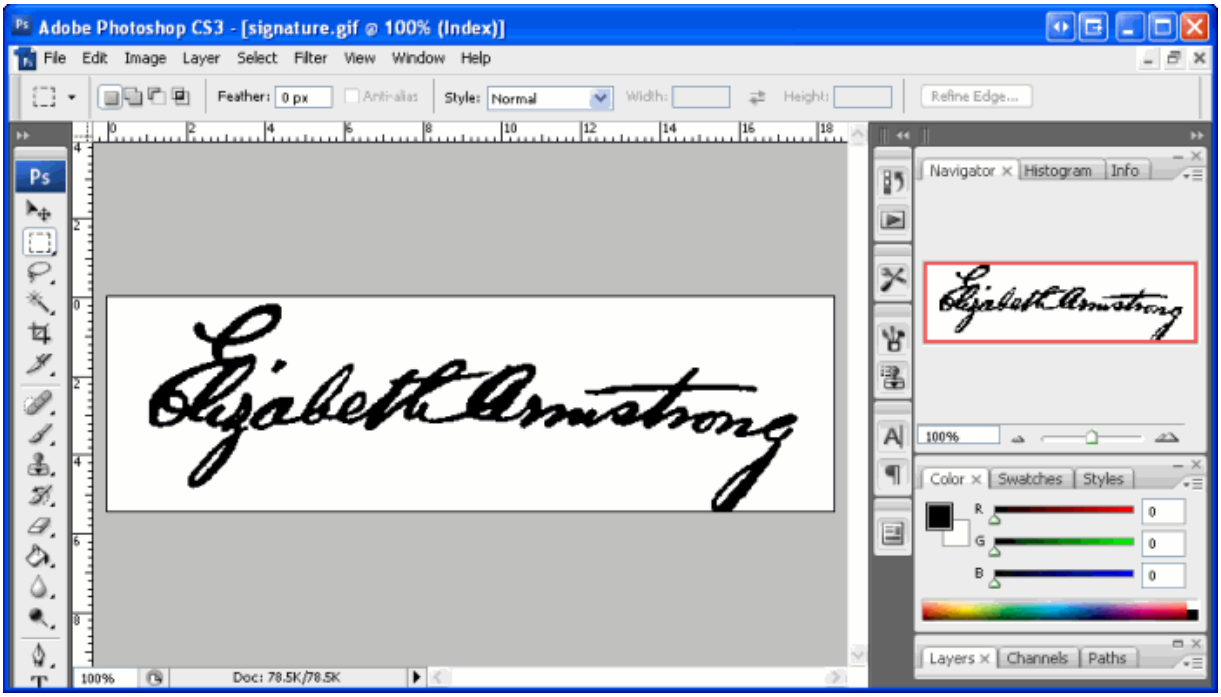
3. Select **View Supplementary**  from the Toolbar beside the *Supplementary* table

   -OR-

   Select **Multimedia>Launch Viewer>Supplementary** in the Menu bar and select the supplementary resource to view:



The selected supplementary resource displays inside the viewer associated with the resource type:

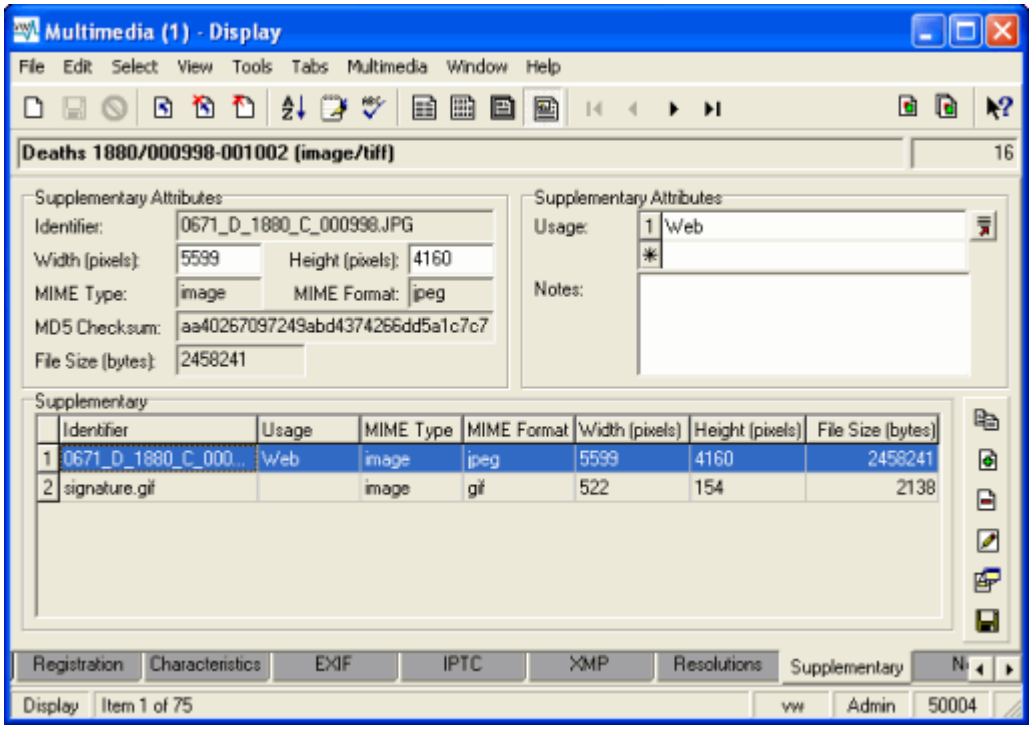


How to save supplementary media

Supplementary media in the *Supplementary* table may be saved to an external location.

In the Multimedia module:

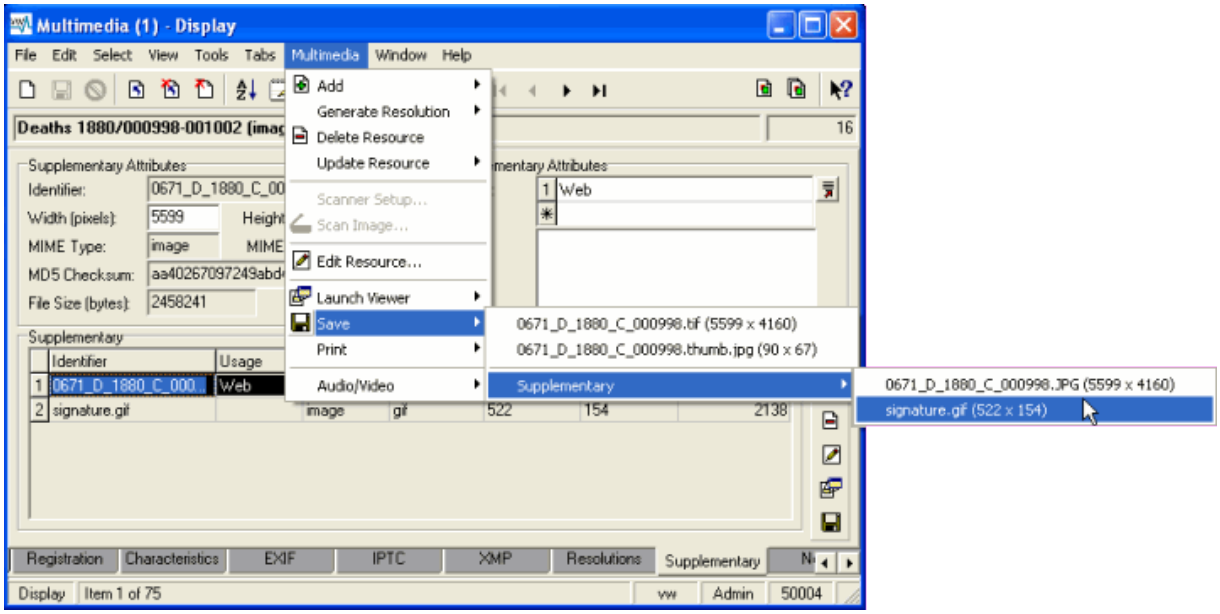
1. Locate the record with the supplementary media to be saved.
2. On the Supplementary tab, click the row in the *Supplementary* table with the media to be saved:



3. Select  from the Toolbar beside the *Supplementary* table

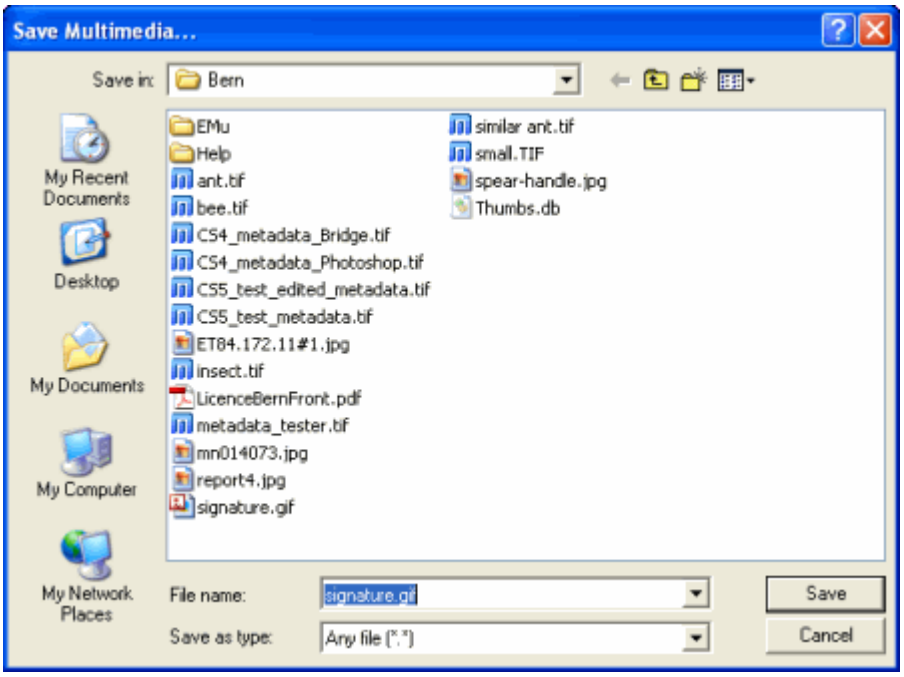
   -OR-

   Select **Multimedia>Save>Supplementary** in the Menu bar and select the supplementary resource to save:



The Save Multimedia dialogue box displays.

4. Navigate to the location where the file is to be saved:



5. Select Save.

The media is saved to the specified file.

How to update supplementary media

Single record

Many of the attributes (e.g. MIME Type, Checksum, etc.) for supplementary media are calculated automatically. As new media is added, the *Supplementary Attributes* fields on the Supplementary tab are maintained by Vitalware.

While it is possible to batch load supplementary media to the Vitalware server, the calculated values are not computed. The Update command recalculates all computed values for all media in the current record's *Supplementary* table.

In the Multimedia module:

1. Locate the record with supplementary media to be updated.
2. Select the **Supplementary** tab.
3. Select **Multimedia>Update Resource>Current Record** in the Menu bar

   -OR-

   Use the keyboard shortcut, `ALT+M+U+C`.

   The Updating Multimedia dialogue box displays:

4. Save the record once the update is complete.

Selected records

The computed values for supplementary media may also be updated for a batch of records. This version of the command is useful after a large import of supplementary media has occurred via the Vitalware server (that is, not imported via the Vitalware client).

In the Multimedia module:

1. Locate the records with supplementary media to be updated.
2. Select **View>List** in the Menu bar

   -OR-

   Select **View List**  in the Toolbar

   -OR-

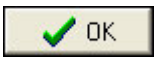
   Use the keyboard shortcut, `ALT+V+I`.
3. Select the records to be updated.

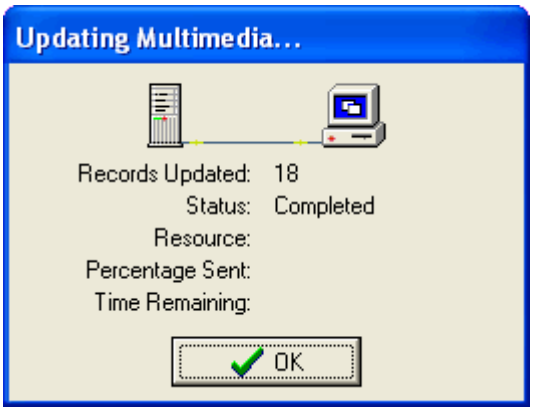
 See Selecting Records in the Vitalware help for more details.

4. Select **Multimedia>Update Resource>Selected Records** in the Menu bar

   -OR-

   Use the keyboard shortcut, `ALT+M+U+S`.

   The Updating Multimedia dialogue box displays.
5. Select  once the update is complete.



SECTION 4

Importing supplementary media

The Vitalware Import tool can be used to import supplementary media files and data into Multimedia records. The mechanism used is the same as for importing multimedia, except that the virtual column *Supplementary_tab* is used to contain the path of the media to be loaded.

The example import file below adds supplementary media to the *Supplementary* table of existing Multimedia records (identified by their IRN):

IRN	Supplementary_tab(+)
1324	E:\Media\Image Text.txt
54765	E:\Media\Cropped Thumbnail.jpg
945632	E:\Media\Audio for 945632.mp3

Vitalware calculates all the computed values and loads them into the appropriate fields as the media is loaded. The identifier assigned to the supplementary media is the file name (without the path) of the imported media.

Multimedia and supplementary media may be loaded in the same import file, as the following import file demonstrates:

MulTitle	Multimedia	Supplementary_tab(1)	Supplementary_tab(2)
Close up of signature.	E:\Media\Sig\Smith_Death_cert.tif	E:\Media\Sig\signature_zoom1.tif	E:\Media\Sig\signature_zoom2.tif
Commemorative Certificate.	E:\Media\Cert\Commemorative2011.tif	E:\Media\Cert\detail.doc	

It is also possible to import *Usage* and *Notes* values along with the media, as the following import file demonstrates:

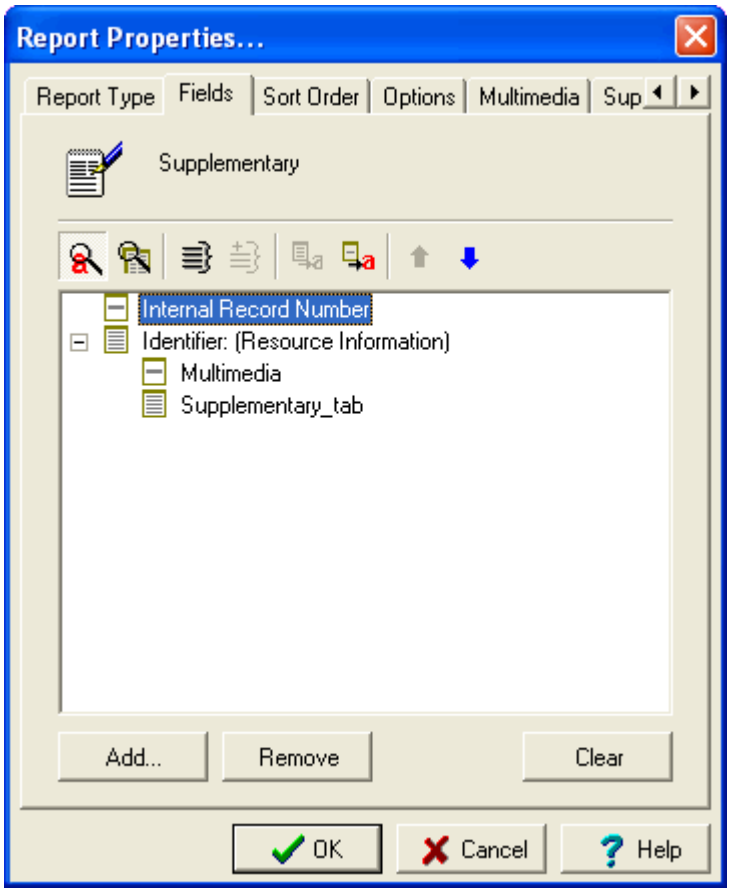
IRN	Supplementary_tab(+, group='import')	SupUsage_nesttab(+, group='import':1)	SupUsage_nesttab(+, group='import':2)	SupNotes0(+, group='import')
1324	E:\Media\Image Text.txt	Web Text		The text in the document contains an English translation of the audio track in the video.
54765	E:\Media\Cropped Thumbnail.jpg	Web Thumbnail	Cropped	A cropped thumbnail of the master image showing the signature only.
945632	E:\Media\Audio for 945632.mp3	Web Audio		An audio description of the commemorative certificates now available for purchase from our website.

Notice how the `group=` import feature is used to ensure the media, usage and notes are all added on the same row in the *Supplementary* table.

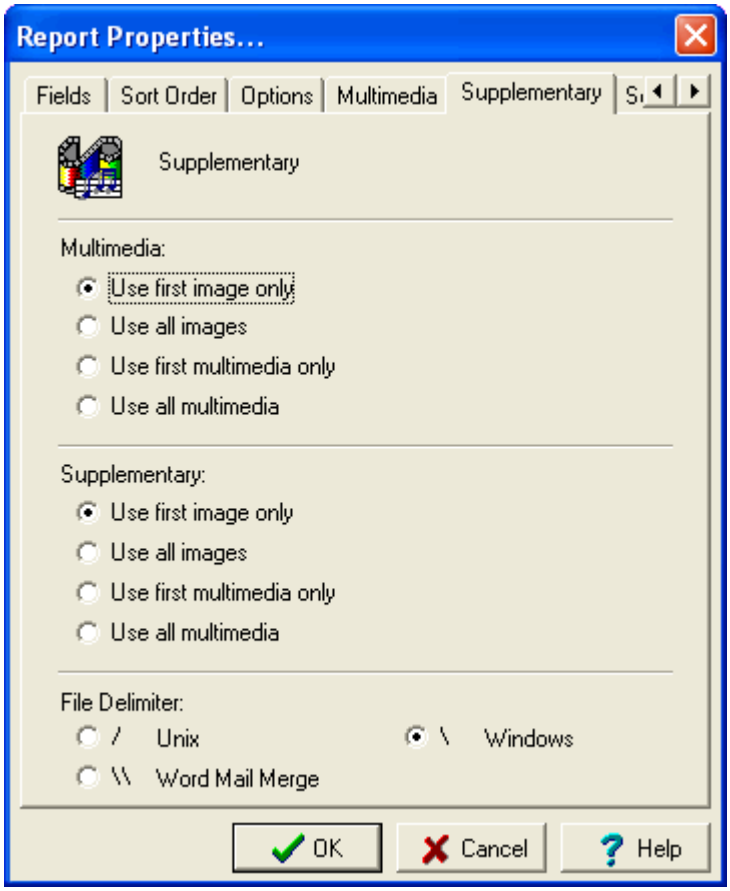
SECTION 5

Reporting with supplementary media

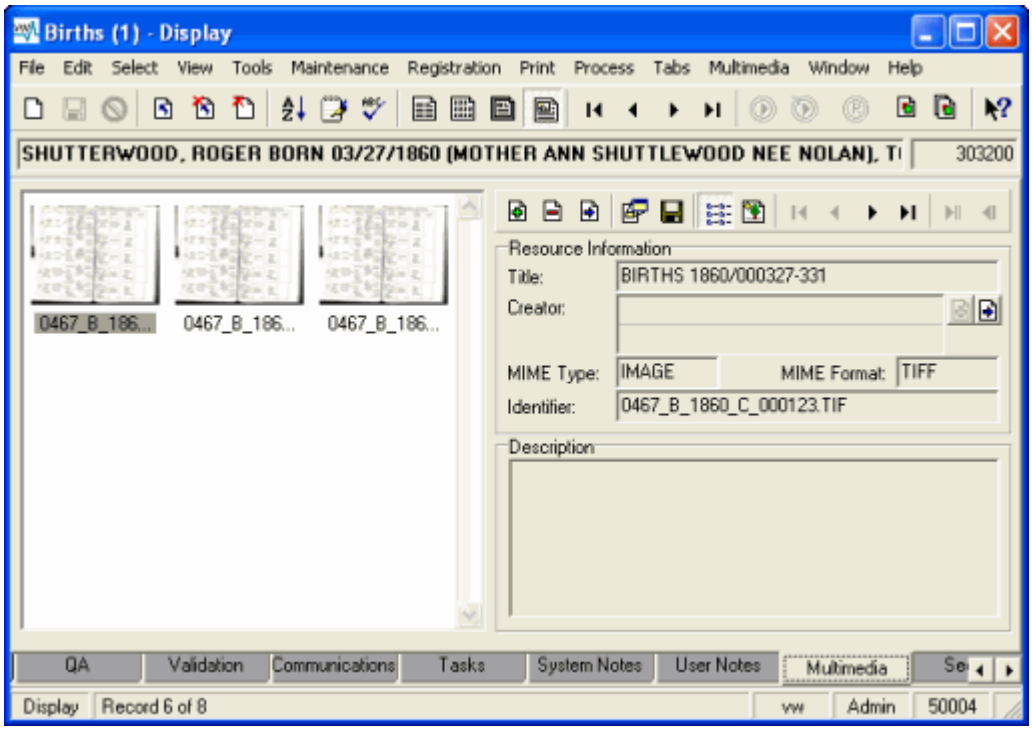
Supplementary media can be included in reports by adding the *Supplementary_tab* column to the list of fields on which to report. The Report Properties dialogue below shows a report containing both the Multimedia and Supplementary media fields (*Multimedia* and *Supplementary_tab* respectively):



When the *Supplementary_tab* column is added to the list of fields on which to report, the Supplementary tab is added to the Report Properties dialogue box. The Supplementary tab provides options that determine which supplementary media is included in the report:



The *Multimedia* set of options controls which Multimedia records attached to a record in a report are included in the report. Consider the Births record below:



A number of Multimedia records are attached to this Births record. The *Multimedia* options determine which of these will be included in the report:

Option	Description
Use first image only	Only the Multimedia record for the first image is included in the report.
Use all images	The Multimedia records for all images are included in the report.
Use first multimedia only	Only the Multimedia record for the first attached record is included in the report.
Use All multimedia	The Multimedia records for all attached records are included in the report.

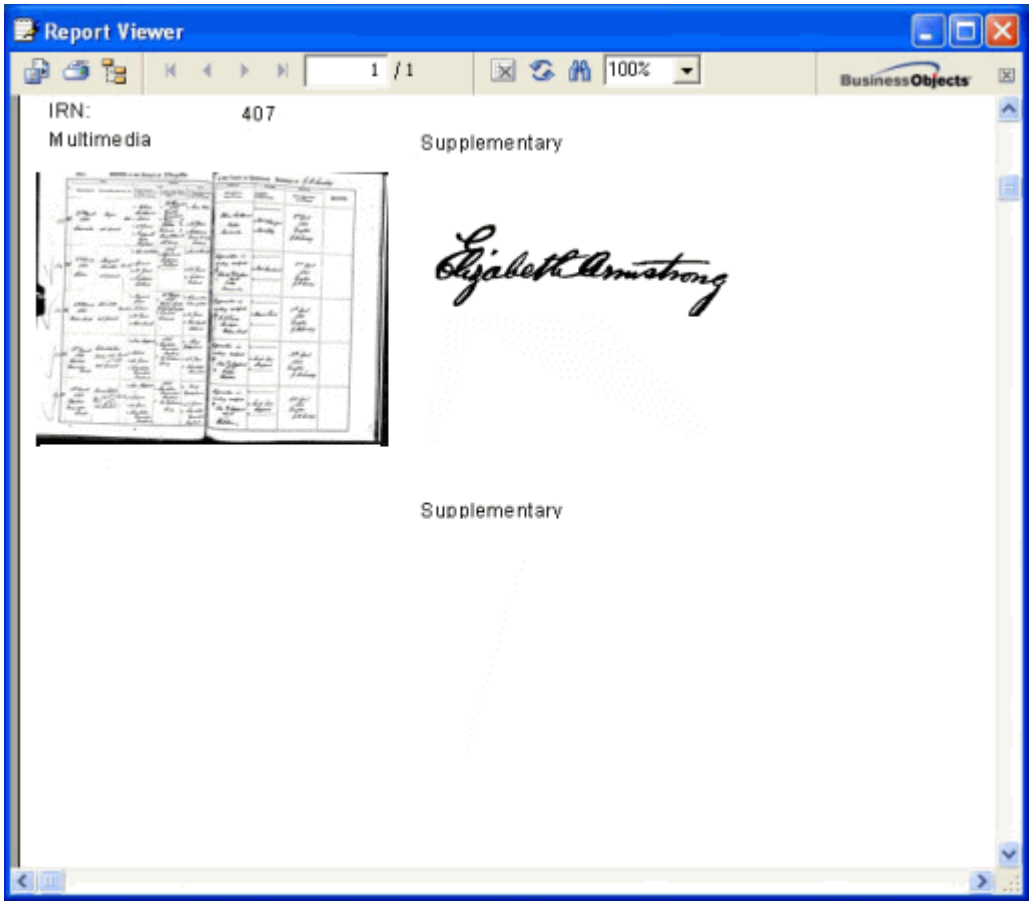
For the Multimedia records specified by the *Multimedia* options, the *Supplementary* options then control which media listed in the *Supplementary* table will be included in the report. The options have the same meaning as in the table above, except that they apply to the *Supplementary* table in the Multimedia record.

The *File Delimiter* option determines how the paths to the media included in the report are to be built:

Option	Description
/ Unix	Use this option if the report is to be placed on a UNIX based computer. This is useful if the output is to be stored on the Vitalware server.
\ *Windows*	Use this option if the report is to be stored on a Windows based computer. In most cases this will be the option required as most reports are displayed on the user's computer.
\\ *Word Mail Merge*	Only use this option if the media is to be included in a Microsoft Word mail merge document.

The report below shows a record with one Multimedia image and one associated supplementary image. In order to produce this output the following options were selected:

Multimedia:	*Use first image only*
Supplementary:	*Use all images*
File Delimiter:	\ *Windows* [This option is used as the report is viewed on the user's Windows computer.]


Report Viewer
1 / 1
100%
BusinessObjects
IRN: 407
Multimedia
Supplementary
Supplementary

SECTION 6

Supplementary media on the Vitalware server

The supplementary media for a Multimedia record is stored in a directory called `supplementary` under the directory in which the master image is stored on the Vitalware server. The location of the master image is determined by the ServerMediaPath Registry entry (or in the absence of this entry, the ServerPath Registry entry). The Registry entry contains a list of paths to consult when locating multimedia on the Vitalware server. The first path in the Registry entry is used to store new multimedia, while all paths are searched to locate multimedia.

There is a special case in which an *exec* entry may be used to interface between Vitalware and a third party imaging system (see Integrating 3rd party imaging systems for more details). In order to support supplementary media while still maintaining the same interface to third party systems, the *filepath* supplied to the `get`, `save`, `ping` and `remove` calls may now include the supplementary directory. For example, the file path passed to the `ping` call for a supplementary image called `Image.jpg` for the Multimedia record with IRN 4254 would be:

```
4/254/supplementary/Image.jpg
```

The `list` call should now return not only all media in the folder passed to it, but all media in the supplementary directory. For example, the call `list 4/254` may result in the following response being returned:

```
Status: success
4/254/master.jpg
4/254/master.thumbnail.jpg
4/254/master.300x300.jpg
4/254/supplementary/video.avi
4/254/supplementary/image.tif
```

Index

Vitalware®
Vital Records Management

The World's Premier Vital Records Software

Vitalware Documentation

Password Management

Document Version 1.1

Vitalware Version 2.2.02

KE Software

www.kesoftware.com

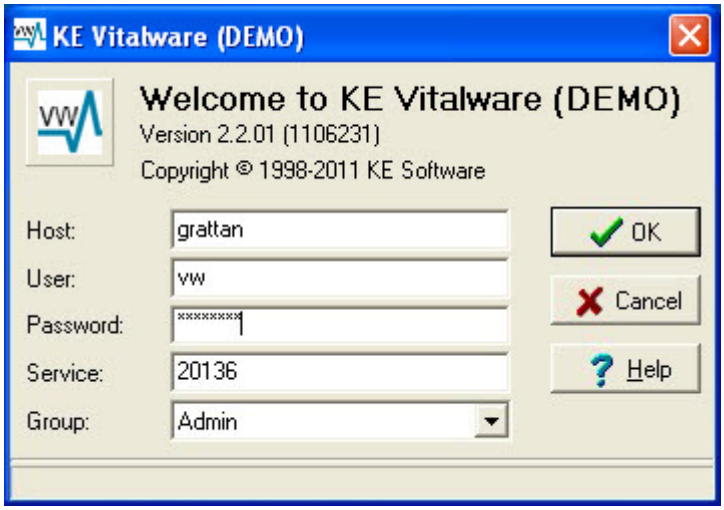
Contents

SECTION 1

Overview

The first step for users using the Windows client to access Vitalware is to log in to the server. The familiar log in dialogue box allows a user to specify their:

- username
- password
- service to which to connect
- group to use (optional)



The information entered into the login dialogue box is transmitted to the Vitalware server for authentication. In particular, the username and password are checked against a database containing user name and password combinations for all persons who are allowed to access the system. Vitalware itself does not store any user name / password pairs but relies on external databases to authenticate users. These external databases are available through a number of sources:

Source	Description
Unix	The traditional Unix user's database consists of a file containing a list of all users who may access the system. The file is located at `/etc/passwd`. A typical entry looks like: `boris:QB2vbP7yzuNzQ:708:400:Boris Badenov:/home/boris:/bin/bash` Each of the fields in the entry is separated by a colon (:). The list of fields is unimportant for this document, except that the first field contains the user name and the second field the password. The password is stored in a one way encrypted format, that is, the password cannot be decrypted (so if you forget your password, your System Administrator cannot tell you what it is and a new password must be set). When a user logs in to Vitalware, their password is encrypted and checked against the encrypted version stored with their user name. If there is a match, the user can then access the system.
Shadow	The Shadow password file is an extension of the base UNIX password file described above. The Shadow file is located in `/etc/shadow`. If a Shadow password file is used, the user's password in `/etc/passwd` is replaced with an `x`. The encrypted password is then stored in the Shadow file. The permissions on the Shadow file only allow the System Administrator account (`root`) to read the contents, thus protecting the encrypted password from general access. The Shadow file contains extra fields used to implement password ageing. A typical entry looks like: `boris:QB2vbP7yzuNzQ:15215:1:30::::` Like the `/etc/passwd` file, the fields are separated by a colon (:) and the first two fields store the user name and the associated encrypted password. The third field stores the number of days between 1 January 1970 and the last time the user changed his password. The next two fields contain the minimum number and maximum number of days between password changes respectively. In the example above, user `boris` must change his password at least every thirty days. If the password is not changed within thirty days, it will expire and a new password must be set next time he logs in to Vitalware. The Shadow password file is available with most versions of Unix, including Solaris and Linux (but not FreeBSD).
NIS	One problem with the Unix and Shadow databases is that they are stored locally. Each machine has its own version of the database, so if a user wants to access more than one machine, an entry needs to exist on each machine for which access is required. A nice solution would be to have a *master* version of the password / Shadow database kept on one machine, and have all other machines contact the master machine when looking up password entries. NIS (Network Information name Service), or YP (Yellow Pages) as it was previously known, provides this functionality. One machine stores the NIS master password / Shadow file and all other machines communicate with the master machine when checking a user name / password combination. NIS can also be used in conjunction with the Unix or Shadow files, providing support for both local users (via Unix and Shadow) and global users (via NIS).

Source	Description
	NIS not only provides password facilities but may also be used to provide a master version of a wide range of other database files. NIS is available for Solaris, Linux and FreeBSD. Due to shortcomings in the original NIS design (e.g. password ageing is not possible), a new version of NIS, known as NIS+, was released. Its purpose is the same as NIS except it is more secure and provides extended user attributes (like password ageing).
LDAP	LDAP (Lightweight Directory Access Protocol) is an extension of the NIS idea. One issue with NIS is that it requires a Unix server. It does not provide a general purpose interface that may be used by non-Unix systems. LDAP addresses this problem by implementing a general purpose database (directory) facility that may be used to store any sort of information (including Unix password information). It then defines a system independent way of looking up this information, thus allowing any type of system to retrieve (and possibly update) data. An explanation of how LDAP is structured is beyond the scope of this document. LDAP is available via the OpenLDAP project for Solaris, Linux and FreeBSD. A number of other implementations are also available. LDAP allows password and Shadow information to be stored in its database, via the `posixAccount` and `shadowAccount` object classes respectively. When a user logs in to Vitalware, the LDAP database is consulted to determine whether access should be granted and determine whether the password has expired. As with NIS, the local password / Shadow files may still be used to store local accounts, while LDAP is used for global accounts.
Windows	Windows provides a local database used to contain user password information (amongst other things). The information is for local accounts only and provides authentication for users accessing the local machine. In this sense it is very similar in functionality to the Unix / Shadow databases (but implemented differently). Windows authentication is available on all versions of Windows.
AD	AD (Active Directory) is the Windows implementation of a general purpose information database (directory). It uses LDAP as one of its access protocols. This means that LDAP may be used to consult the Active Directory database. Active Directory allows password / Shadow information to be stored and retrieved via the `posixAccount` and `shadowAccount` object classes respectively. As with LDAP the information stored is for global accounts. AD is provided with Windows Server systems and can be queried by Solaris, Linux, FreeBSD and Windows. The local Windows accounts may still be used to register local users, while AD is used for global accounts.

As you can see, there are a number of alternatives available for registering Vitalware users and their passwords. In general, each institution will have a policy dictating which of the above sources should be used for user authentication.

Vitalware implements three mechanisms for determining whether a user name / password combination is correct. The mechanisms are:

Mechanism	Description
PAM	The Pluggable Authentication Module, or PAM for short, is available on all version of Unix, including Solaris, Linux and FreeBSD. It is not available on Windows servers. PAM is a flexible mechanism that uses a configuration file to determine what password sources should be consulted to retrieve password / Shadow information. It provides support for the following database sources: • Unix • Shadow • NIS/NIS+ • LDAP • AD PAM is the most common look-up mechanism used by Vitalware where the Vitalware server is installed on a Unix system.
SFU / SUA	Services For Unix (SFU), or Subsystem for UNIX-based Applications (SUA) as it is now known, provides password authentication on Windows based Vitalware servers. SFU / SUA is not configurable. It provides support for the following sources: • Windows • AD SFU / SUA is the look-up mechanism used by Vitalware on Windows based Vitalware servers.
Traditional	If a Unix system does not provide support for PAM, then the Traditional look-up mechanism is used. The Traditional system provides support for the following sources: • Unix • Shadow Traditional support is only provided where institutions elect to not use PAM support. In other words, very rarely!

New Features

Now we have all the theory out of the way, let's look at the new features added to Vitalware to provide support for password management:

New Feature	Description
Change password	A user may change their password from within the Vitalware Windows client. The new password is checked to see if it conforms to minimum standards (e.g. length, character mix, etc.) before being set.
Password expiry	A user's password may expire. An expired password occurs when a user has not changed their password within a prescribed number of days. Once the password expires, the user must set a new password the next time they log in to Vitalware. They will be prompted for the new password.
Expire account	A user's account may expire. Once the account has expired the user will no longer be able to access Vitalware. There are two ways to expire an account: • The user has not logged in to Vitalware for a given number of days. • An absolute date after which access to the system will be denied.
Reset password	If a user forgets their password, the System Administrator can clear the old password (or set a new one) and force the user to enter a new password the next time they log in to Vitalware.
Lock account	The System Administrator may lock a user's account. While the account is locked the user will not be able to access Vitalware. Once the account is unlocked the user may use Vitalware once again. Account locking is useful if someone is leaving for an extended period of time, but may return some time in the future.

The implementation of password management in Vitalware 2.2.02 provides System Administrators with a range of options in terms of managing user authentication and ageing passwords. It also allows users to change their passwords from within the Vitalware Windows client.

SECTION 2

Using Password Management

The majority of the new password management functionality is available in Vitalware 2.2.02 without the need to have it enabled. In fact, most of the client side functionality is invoked by requests from the Vitalware server. For example, if a user's password has expired, the Vitalware server will inform the Vitalware Windows client next time the user logs in to Vitalware. At this time the user will be prompted for a new password.

The only feature that is instigated by the user is the ability to change their password. In order to change a user's password, the Vitalware server must provide the required support. Both PAM and SFU / SUA provide mechanisms for updating a password, whereas Traditional does not. It is also possible to configure PAM to not support password updates, or to add in other authentication mechanisms (e.g. dongles) that do not provide password updates.

The Vitalware client cannot determine what level of support is provided by the Vitalware server for password changing. As such, a new Registry entry has been added to indicate whether password changing is supported. The format of the entry is:

```
User|user|Setting|Change Password|value
Group|group|Setting|Change Password|value
System|Setting|Change Password|value
```

where:

value is either `true` or `false`.

A `true` value indicates that password changing is supported, while a `false` value removes the Change Password command from the Tools menu in the Windows client. The default value is `true`. Thus, if you are using the Traditional password mechanism, you will need to disable password changing explicitly.

1. Changing your Password

A user may change their password provided the Change Password Registry entry is not set to `false`.

Server support required:

- PAM
- SFU / SUA

In Vitalware:

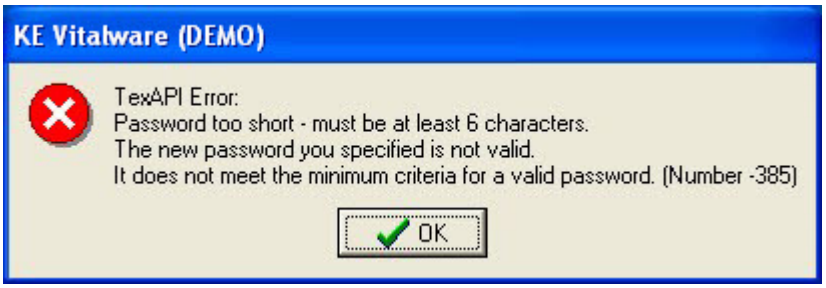
1. Open any module.
2. Select **Tools>Change Password...** in the Menu bar
   -OR-
   Use the keyboard shortcut, `ALT+T+H`.
   The Change Password dialogue displays:

3. Enter your existing password and your new password twice.
   Both instances of the new password must be the same.
4. Click 

   .
   The password is updated on the server.

If the two instances of the new password do not match, an error message displays:

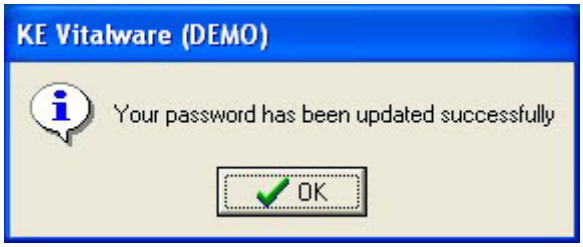
If the new password does not pass the validation criteria, an error message displays:



In both of the above cases the Change Password dialogue displays allowing the error to be corrected.

5. Click 

once the password is updated:



2. Updating an Expired Password

A user's password may expire for one of two reasons:

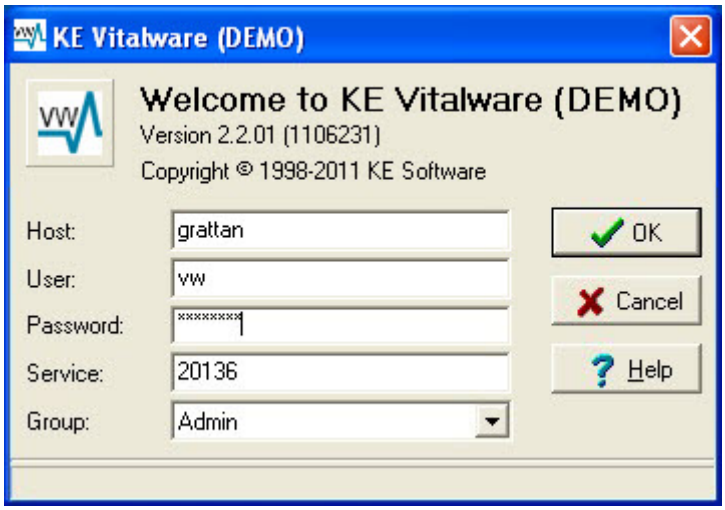
- The System Administrator has expired the password.
- The user has not updated their password within a given time frame. The time frame for password updates is set on the Vitalware server and can vary from institution to institution.

Server support required:

- PAM

In Vitalware:

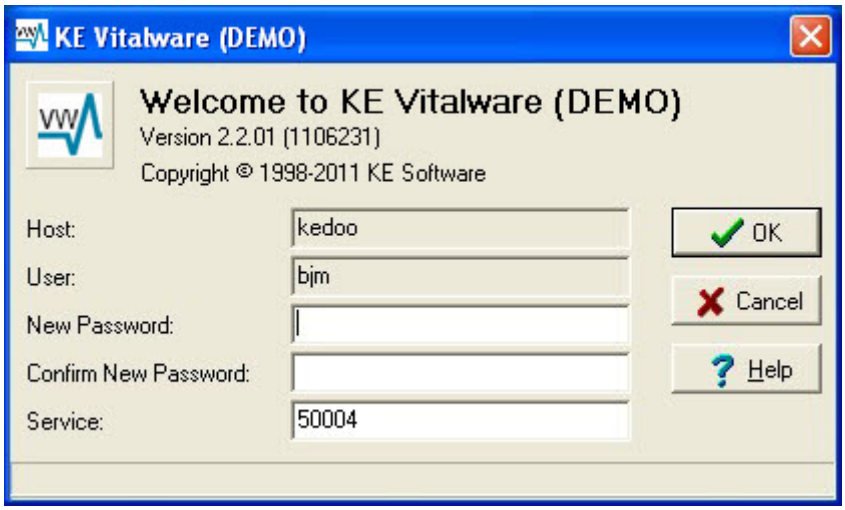
1. Start up Vitalware.
   The log in dialogue displays:



6. Enter your log in details and click  .
   If your password has expired, a message displays:

7. Click  .

A new log in dialogue displays:



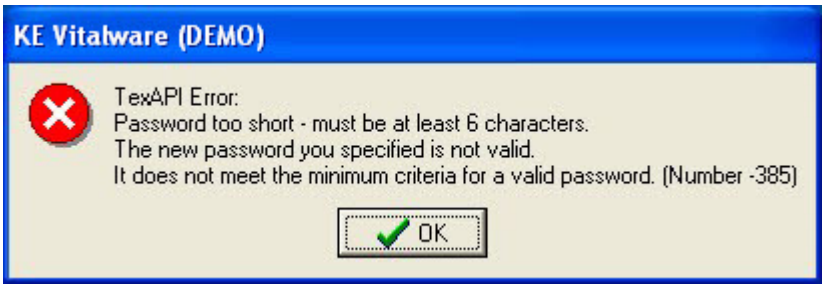
8. Enter a new password and confirm it by re-entering it.
9. Click 

.

The password is updated on the server.

If the two instances of the new password do not match, an error message displays:

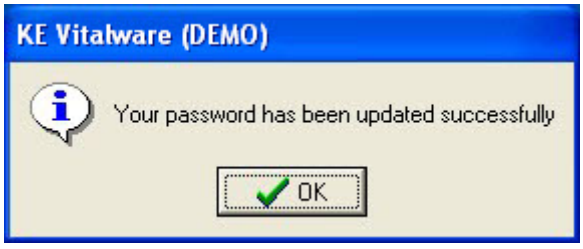
If the new password does not pass the validation criteria, an error message displays:



In both cases the log in dialogue is displayed allowing the error to be corrected.

10. Click 

once the password is updated:



3. Password Admin Task

A user may also change their password via an Admin task. The task is provided for systems that do not support password changes via PAM or SFU/SUA. The following Registry entry is required to provide the Change Password Admin task:

```
Group|Default|Table|eadmin|Admin Task|Change Password|password
'[Password:Old Password]' '[Password:New Password]'
'[Password:Confirm New Password]'
```

Once the Registry entry is specified, the task becomes available.

Server support required:

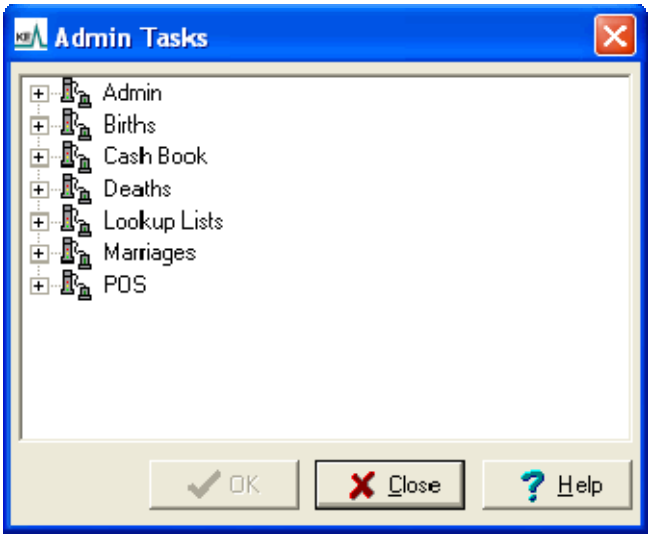
- PAM
- Traditional

In Vitalware:

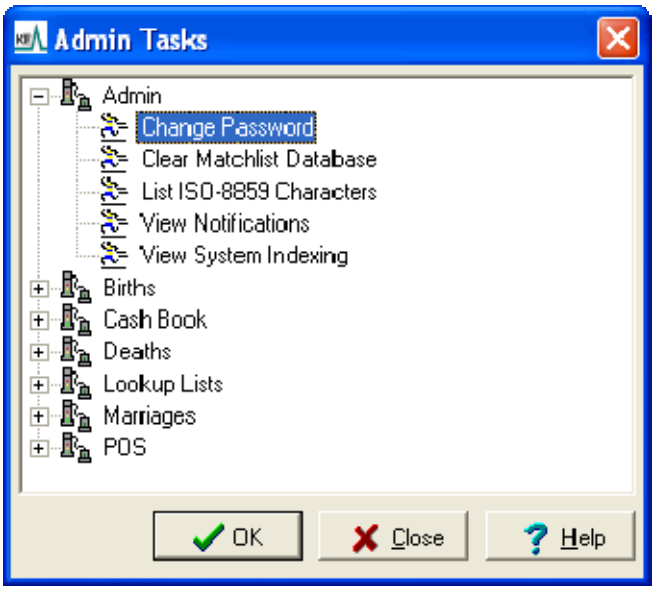
1. Select Admin 

from the Command Centre.

The Admin Tasks dialogue displays.



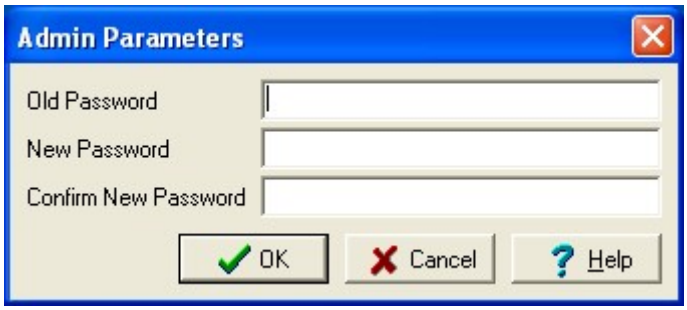
11. Expand the Admin node:



12. Select **Change Password** and click 

.

The Admin Parameters dialogue displays:



13. Enter your existing password.
14. Enter a new password and confirm it by re-entering it.
15. Click 

.

The password is updated on the server.

If the two instances of the new password do not match, an error message displays:

Re-enter the new password.

If the new password does not pass the validation criteria, an error message displays:

Enter a new password that passes validation.

16. Click  once the password is updated:

SECTION 3

Managing Passwords

In this section we look at the commands used by a System Administrator to manage user passwords. Unfortunately there is no utility common to all systems on which Vitalware runs (Unix and Windows) that provides a common interface to password management. In fact, even within the Unix family of systems no such utility exists. As such, we will look at the support provided by the four most common platforms on which the Vitalware server is installed:

- Solaris 10
- Linux
- FreeBSD
- Windows

1. Password Ageing

Password ageing allows a System Administrator to force users to change their password within a given number of days. For example, your institution may have a policy that users must change their passwords at least once a quarter.

Solaris 10

The `passwd` command is used to set up password ageing on Solaris 10. The format of the command is:

```
passwd -n min -x max user
```

where:

min is the minimum number of days required between password changes.

Not supported by Vitalware due to limitations in Solaris.

max is the maximum number of days for which the password is valid. Once *max* days have elapsed without a password change, the user will be prompted for a new password at the next successful log in. A *max* value of `-1` is used to disable password ageing.

user is the name of the user account to which the restrictions are to apply.

The default values for *min* and *max* are defined in the file `/etc/default/passwd`. There are two variables used to set the minimum and maximum ageing values:

- `MINWEEKS`

  The minimum number of weeks between password changes. The default value is empty, implying no minimum is set.

  Not supported by Vitalware due to limitations in Solaris.

- `MAXWEEKS`

  The maximum number of weeks between password changes. The default value is empty, implying no maximum is set.

Setting the minimum and / or maximum default values to non-empty in `/etc/default/passwd` will result in users without ageing having it enabled the next time their password is modified.

Password ageing is supported by the Shadow and NIS+ password databases. LDAP and AD also support password ageing via the `shadowAccount` class object.

Linux

The `chage` command is used to set up password ageing on Linux. The format of the command is:

```
chage -m min -M max user
```

where:

min is the minimum number of days required between password changes. A value of zero indicate there is no minimum.

max is the maximum number of days for which the password is valid. Once *max* days have passed without a password change, the user will be prompted for a new password at the next successful log in. A `max` value of `99999` is used to disable password ageing.

user is the name of the user account to which the restrictions are to apply.

The default values for *min* and *max* are defined in the file `/etc/login.defs`. There are two variables used to set the minimum and maximum ageing values:

- `PASS_MIN_DAYS`

  The minimum number of days between password changes. The default value is zero, implying no minimum is set.

- `PASS_MAX_DAYS`

  The maximum number of days between password changes. The default value is `99999`, implying no maximum is set.

Setting the minimum and / or maximum default values to non-empty in `/etc/login.defs` will result in users without ageing having it enabled the next time their password is modified.

Password ageing is supported by the Shadow password database only. LDAP and AD also support password ageing via the `shadowAccount` class object, however changes to a user's settings must be made via `ldapmodify` or an ldap client (e.g. Active Directory Explorer for AD).

FreeBSD

FreeBSD provides a limited form of password ageing. Rather than setting a minimum and maximum number of days between password changes, it allows you to set the date on which a password should expire. Once the date arrives the user will be prompted for a new password. The `pw` command is used to set the date on which a password expires:

```
pw usermod user -p date
```

where:

user is the name of the user account to which the restrictions are to apply.

date is the date on which the password will expire. The date format is `dd-mmm-yyyy` (e.g. `23-Oct-2011`). An empty value is used to remove an expiry date.

When the new password is set the expiry date field is cleared.

If you want to have a new expiry date calculated automatically when a password is set, you need to specify the `passwordtime` attribute in the log in class file located at `/etc/login.conf`. The login class file allows a set of system attributes (resource usage, etc.) to be set on a login class basis. A user is then assigned to a login class using the `pw` command:

```
pw usermod user -L class
```

where:

user is the name of the user account to be added to the login class.

class is the class name as specified in the file `/etc/login.conf`.

To set the system wide password expiry date, the *default* login class should be modified to:

```
default:\
       :passwordtime=time:\
       ...
```

where:

time is the time interval to set for a password to expire. A large number of formats are available for the value with `nnnd` being the most common. For example, `90d` would indicate the password will expire ninety days after it was last set.

If you change values in `/etc/login.conf`, you need to rebuild the internal database by executing:

```
cap_mkdb /etc/login.conf
```

Password ageing is supported by Unix and NIS+ password databases only. LDAP and AD also support password ageing via the `shadowAccount` class object, however changes to a user's settings must be made via `ldapmodify` or an ldap client (e.g. Active Directory Explorer for AD).

Windows

Password ageing is set on a Windows server running Vitalware via either a Local or Global Security Policy. The Local Security Policy editor is invoked by running `secpol.msc`. The Global Security Policy Editor is started by running `gpedit.msc`. The policy paths are:

- Local Security Policy
  `Security Settings/Account Policies/Password Policy`
- Global Security Policy
  [computer name] `Policy/Computer Configuration/Windows Settings/Security Settings/Account Policies/Password Policy`

There are two attributes used to set the minimum and maximum ageing values:

- *Minimum password age*
  The minimum number of days between password changes. A value of zero implies there is no minimum number of days.
- *Maximum password age*
  The maximum number of days between password changes. A value of zero implies there is no maximum number of days.

When a password expires, the user will be prompted to enter a new password when they next log in to Windows. Vitalware will not prompt for a new password when a Windows password expires, rather the user will not be able to access the system.

Examples

Example 1

Our institution has a policy that passwords must be changed at least once a quarter. There is no minimum time between changes.

Solaris 10

As the policy is a default setting the best solution is to edit `/etc/default/passwd` and set the following entries:

```
MINWEEKS=
MAXWEEKS=13
```

Linux

As the policy is a default setting the best solution is to edit `/etc/login.defs` and set the following entries:

```
PASS_MIN_DAYS=0
PASS_MAX_DAYS=90
```

FreeBSD

As the policy is a default setting the best solution is to edit `/etc/login.conf` and update the default login class to contain the following entry:

```
default:\
       :passwordtime=90d:\
       ...
```

Windows

If the setting is to be domain wide, then the Global Security Policy should be updated, otherwise if it is machine specific, the Local Security Policy should be used. The following attributes should be set:

`Minimum password age` set to `0`.

`Maximum password age` set to `90`.

Example 2

We have a default policy of forcing password changes every quarter, however we have a certain user (`boris`) for which we would like to disable password ageing.

Solaris 10

The following command may be used to disable password ageing for user `boris`:

```
passwd -x -1 boris
```

Linux

The following command may be used to disable password ageing for user `boris`:

```
chage -M 99999 boris
```

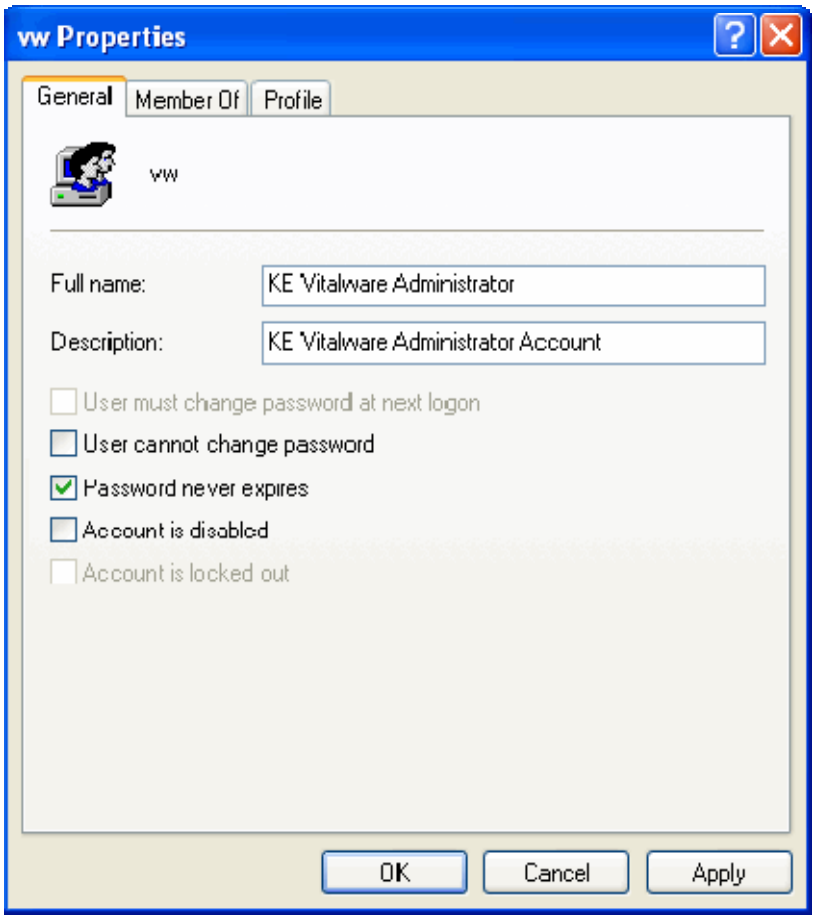
FreeBSD

The following command may be used to disable password ageing for user `boris`:

```
pw usermod boris -p ''
```

Windows

View the properties for the user account. If Active Directory is enabled, run `dsa.msc` to view Active Directory users, otherwise run `lusrmgr.msc` to list local users. Right-click on user `boris` and select **Properties**. Turn on the *Password never expires* checkbox:


vw Properties
General
Member Of
Profile
vw
Full name:
KE Vitalware Administrator
Description:
KE Vitalware Administrator Account
User must change password at next logon
User cannot change password
Password never expires
Account is disabled
Account is locked out
OK
Cancel
Apply

2. Password Reset

The password reset facility allows a System Administrator to force a user to change their password the next time they log in successfully. It is generally used when a user forgets their password. In this case the System Administrator sets a new password and then forces the user to change it the next time they access the system.

Solaris 10

The `passwd` command is used to force a user to change their password the next time they log in to Vitalware. The format of the command is:

```
passwd -f user
```

where:

user is the name of the user account to reset.

Password resetting is supported by the Shadow and NIS+ password databases. LDAP and AD also support password resetting via the `shadowAccount` class object.

Linux

The `chage` command is used to force a user to change their password the next time they log in to Vitalware. The format of the command is:

```
chage -d 0 user
```

where:

user is the name of the user account to reset.

Password resetting is supported by the Shadow and NIS+ password databases. LDAP and AD also support password resetting via the `shadowAccount` class object.

FreeBSD

The `pw` command is used to force a user to change their password the next time they log in to Vitalware. The format of the command is:

```
pw usermod user -p 01-jan-2000
```

where:

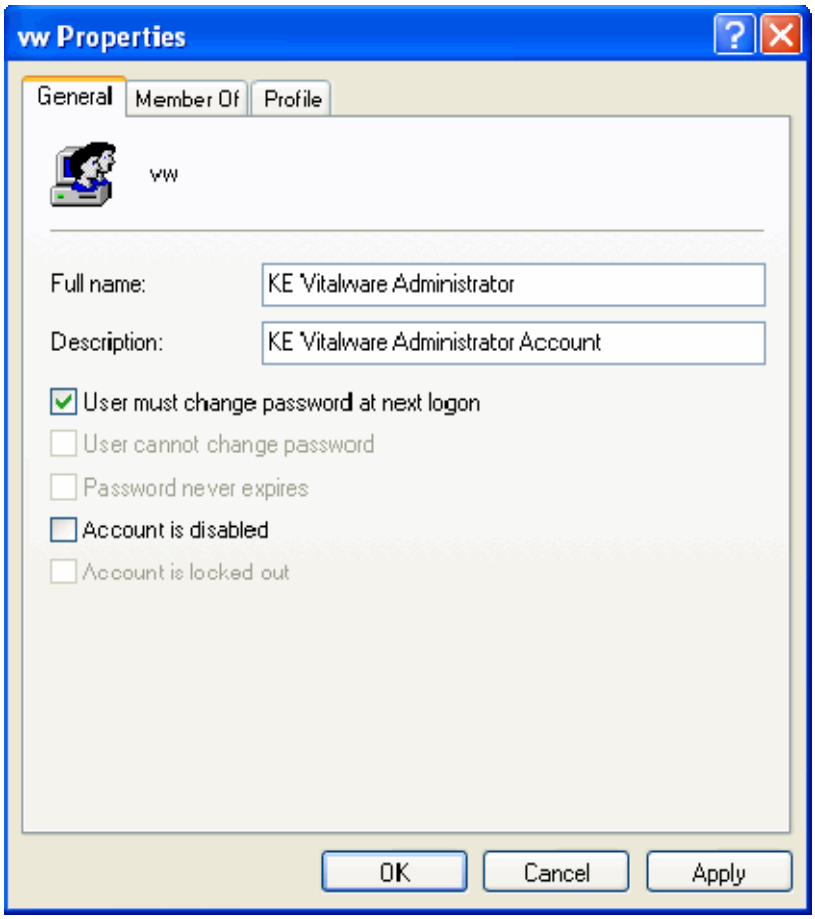
user is the name of the user account to reset.

Password resetting is supported by the Shadow and NIS+ password databases. LDAP and AD also support password resetting via the `shadowAccount` class object.

Windows

View the properties for the user account. If Active Directory is enabled, run `dsa.msc` to

view the Active Directory users, otherwise run `lusrmgr.msc` to list local users. Turn off the *Password never expires* checkbox, then turn on the *User must change password at next logon* checkbox:



Example

User `boris` has forgotten his password. We would like to reset it to his user name and force him to change it the next time he logs in to Vitalware.

Solaris 10

The commands required to reset `boris`'s password are:

```
passwd boris
passwd -f boris
```

Linux

The commands required to reset `boris`'s password are:

```
passwd boris
chage -d 0 boris
```

FreeBSD

The commands required to reset `boris`'s password are:

```
passwd boris
pw usermod boris -p 01-jan-2000
```

Windows

If Active Directory is enabled, run `dsa.msc` to view the Active Directory users, otherwise run `lusrmgr.msc` to list local users. Right-click on user `boris` and select **Set Password....** Change the password as required. View the properties for user `boris` and turn off the *Password never expires* checkbox, then turn on the *User must change password at next logon* checkbox.

3. Account Ageing

The account ageing facility allows a System Administrator to set a date after which the user account becomes inactive. If a user tries to access Vitalware after the set date, a message indicating that the account has expired is displayed and access is denied.

Solaris 10

The `usermod` command is used to set / remove an expiry date for a user account. The format of the command is:

```
usermod -e date user
```

where:

- date — is the date after which the account is no longer valid. The date format used is `mm/dd/yy`. An empty date value is used to remove an expiry date.
- *user* — is the name of the user account to expire.

Account ageing is supported by the Shadow and NIS+ password databases. LDAP and AD also support account expiry via the `shadowAccount` class object, however setting the expiry date must be done via an LDAP client, rather than the `usermod` command.

Linux

The `chage` command is used to set / remove an expiry date for a user account. The format of the command is:

```
chage -E date user
```

where:

- *date* — is the date after which the account is no longer valid. The date format used is `yyyy-mm-dd`. An empty date value is used to remove an expiry date.
- *user* — is the name of the user account to expire.

Account ageing is supported by the Shadow and NIS+ password databases. LDAP and AD also support account expiry via the `shadowAccount` class object, however setting the expiry date must be done via an LDAP client, rather than the `chage` command.

FreeBSD

The `pw` command is used to set / remove an expiry date for a user account. The format of the command is:

```
pw usermod user -e date
```

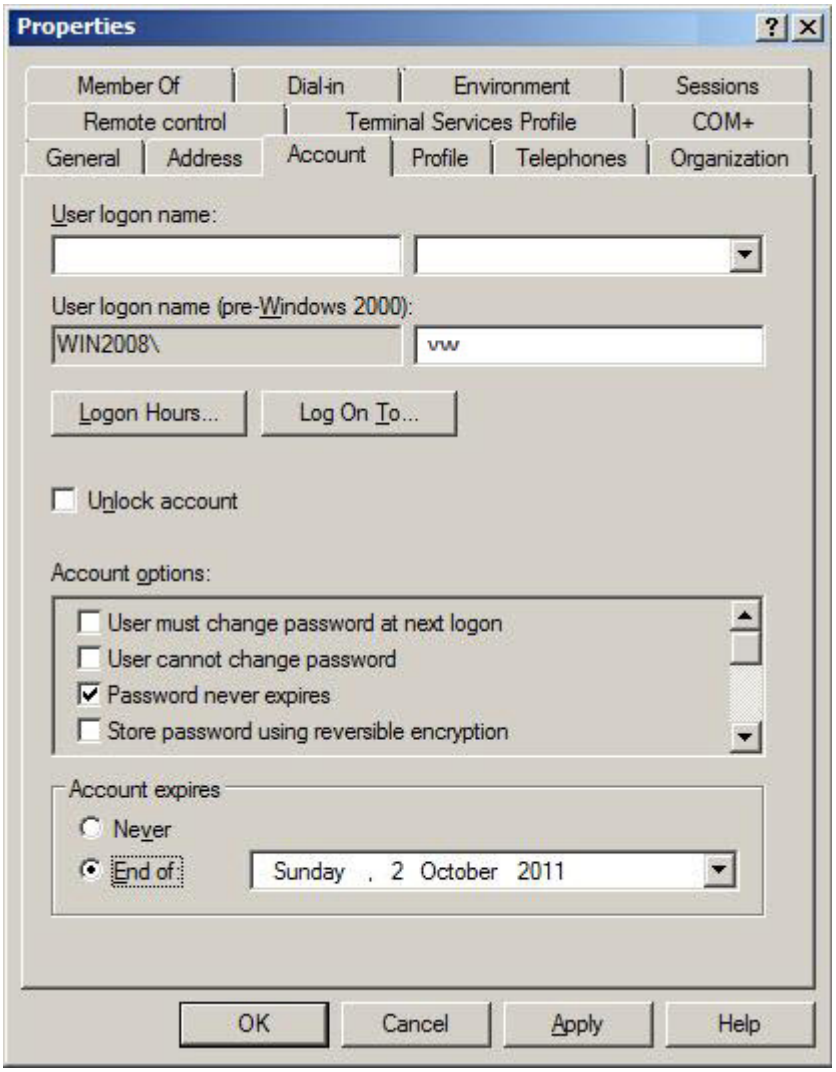
where:

date	is the date after which the account is no longer valid. The date format used is `dd-mm-yyyy`. An empty date value is used to remove an expiry date.
user	is the name of the user account to expire.

Account ageing is supported by the Unix and NIS+ password databases. LDAP and AD also support account expiry via the `shadowAccount` class object, however setting the expiry date must be done via an LDAP client, rather than the `pw` command.

Windows

Account ageing is only supported for Active Directory user accounts. Local accounts cannot have an expiry date. Run `dsa.msc` and view the properties for the user account. On the *Account* tab turn on the *End of* radio button in the *Account expires* group box. Set the date after which the account will not be active:



Examples

Example 1

We have a number of students working for our institution over the summer break. We would like to expire their accounts once they return to university on 1st March 2012.

In each case below, it is not possible to set the expiry date for all students with the one command. Each user account needs to be set individually.

Solaris 10

The command required to set the expiry date is:

```
usermod -e '03/01/12' student1
```

Linux

The command required to set the expiry date is:

```
chage -E '2012-03-01' student1
```

FreeBSD

The command required to set the expiry date is:

```
pw usermod student1 -e '01-03-2012'
```

Windows

Run `dsa.msc` and view the properties for the `student1` user account. On the Account tab turn on the *End of* radio button in the *Account expires* group box. Set the expiry date to `1 March 2012`.

Example 2

One of the students (`student2`) is not returning to university and will continue to work for us for the foreseeable future. We need to remove their expiry date.

Solaris 10

The command required to remove the expiry date is:

```
usermod -e '' student2
```

Linux

The command required to remove the expiry date is:

```
chage -E '' student2
```

FreeBSD

The command required to remove the expiry date is:

```
pw usermod student2 -e ''
```

Windows

Run `dsa.msc` and view the properties for the `student2` user account. On the Account tab turn on the *Never* radio button in the *Account expires* group box.

4. Account Locking

The account locking facility allows a System Administrator to turn off access for a user account. While the account is locked, the user will not be able to log in to Vitalware. The error message will indicate an incorrect user name / password combination has been supplied. When an account is unlocked the user may access Vitalware again using their old password. Account locking is useful if a user is leaving for an extended period of time, but does plan to return in the future.

Solaris 10

The `passwd` command is used to lock / unlock a user's account. The format of the commands used to lock and unlock an account respectively is:

```
passwd -l user
passwd -u user
```

where:

user is the name of the user account to lock / unlock.

Account locking is supported by the Shadow and NIS+ password databases. LDAP and AD also support account locking via the `posixAccount` class object.

Linux

The `passwd` command is used to lock / unlock a user's account. The format of the commands used to lock and unlock an account respectively is:

```
passwd -l user
passwd -u user
```

where:

user is the name of the user account to lock / unlock.

Account locking is supported by the Shadow and NIS+ password databases. LDAP and AD also support account locking via the `posixAccount` class object.

FreeBSD

The `pw` command is used to lock / unlock a user's account. The format of the commands used to lock and unlock an account respectively is:

```
pw lock user
pw unlock user
```

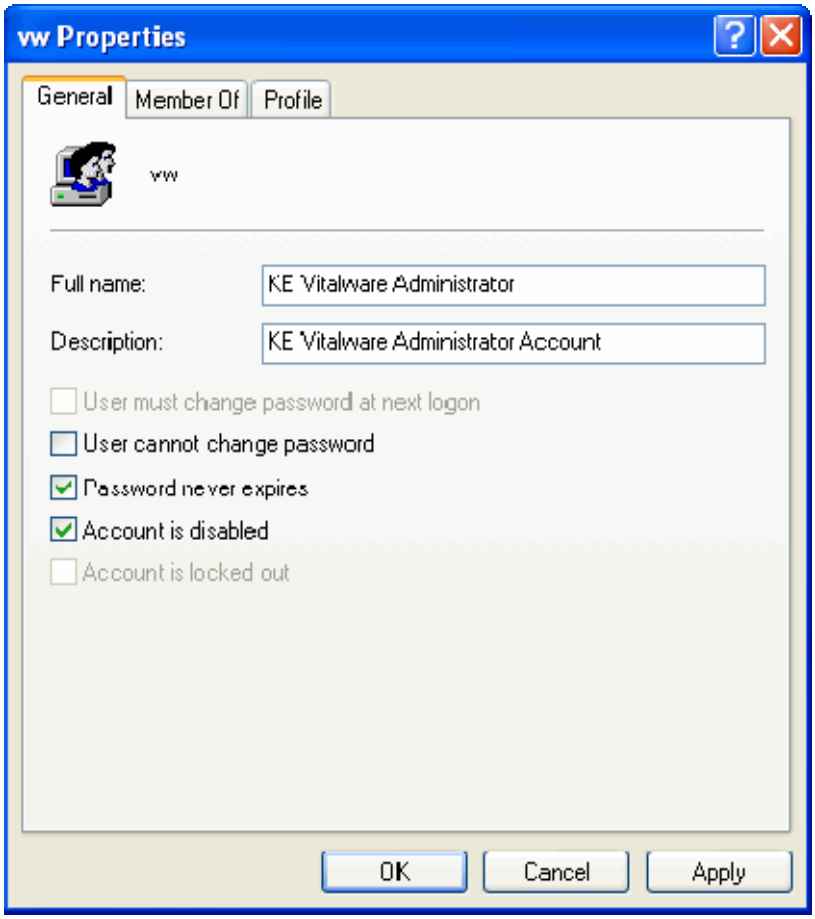
where:

user is the name of the user account to lock / unlock.

Account locking is supported by the Unix and NIS+ password databases. LDAP and AD also support account locking via the `posixAccount` class object.

Windows

View the properties for the user account. If Active Directory is enabled, run `dsa.msc` to view the Active Directory users, otherwise run `lusrmgr.msc` to list local users. Turn on the *Account is disabled* checkbox:



Examples

Example 1

User `boris` has taken leave for a year. We would like to lock his account while he is away.

Solaris 10

The command required to lock the user account is:

```
passwd -l boris
```

Linux

The command required to lock the user account is:

```
passwd -l boris
```

FreeBSD

The command required to lock the user account is:

```
pw lock boris
```

Windows

If Active Directory is enabled, run `dsa.msc` to view the Active Directory users, otherwise run `lusrmgr.msc` to list local users. View the properties for user account `boris`. Turn on the *Account is disabled* checkbox.

Example 2

User `boris` has now returned from leave and we would like to unlock his account.

Solaris 10

The command required to unlock the user account is:

```
passwd -u boris
```

Linux

The command required to unlock the user account is:

```
passwd -u boris
```

FreeBSD

The command required to unlock the user account is:

```
pw unlock boris
```

Windows

If Active Directory is enabled, run `dsa.msc` to view the Active Directory users, otherwise run `lusrmgr.msc` to list local users. View the properties for user account `boris`. Turn off the *Account is disabled* checkbox.

5. Maximum Retries

The maximum retries setting allows a System Administrator to set the maximum number of incorrect passwords that may be entered consecutively before the user's account is locked. Implementing a maximum retries limit is useful if you suspect people are trying to break into user accounts by guessing their password.

Solaris 10

Solaris 10 provides a variable called `RETRIES` in the file `/etc/default/login` for setting the maximum number of password retries before an account may be locked. The default value is empty, implying no limit.

Solaris 10, by default, does not lock an account when the limit is reached. In order to lock the account two mechanisms are provided. The first mechanism is system wide, while the second is on a per user basis. The system wide setting is found in the file `/etc/security/policy.conf`. The `LOCK_AFTER_RETRIES` variable must be set to `YES` to force accounts to be locked once the maximum number of retries is triggered. The setting may also be enabled on a per-user basis in the file `/etc/user_attr`. The `usermod` command is used to set the per-user value:

```
usermod -K lock_after_retries=value user
```

where:

value is `no` (account locking is disabled once the retries limit is reached) or `yes` (account locking is enabled once the retries limit is reached).

user is the name of the user account for which to set `lock_after_retries`.

Once an account has been locked due to exceeding the maximum number of tries, the user cannot log in to Vitalware until the account is unlocked (via `passwd -u` *user*).

Account locking after a maximum number of incorrect passwords is supported by the Shadow and NIS+ password databases. LDAP and AD also support account locking via the `posixAccount` and `shadowAccount` class objects.

Linux

Linux uses the `pam_tally` module in the PAM sub-system to implement account locking after a set number of failed passwords. In order to enable the module you need to include it in the Texpress PAM configuration file located at `/etc/pam.d/texpress`. The following two entries need to be added:

```
auth      required     pam_tally.so onerr=fail deny=5
account   required     pam_tally.so reset
```

The order of these two lines in the PAM configuration file is important. The first line must appear **before** the entry for:

```
auth ... pam_unix.so
```

and the second line must appear **after** the entry for:

```
account ... pam_unix.so
```

The `deny` argument determines the number of failed password attempts before the user can no longer log in to the system.

Once a user has reached the `deny` limit, they cannot log in until the number of failed passwords is reset to zero. The `faillog` command is used to reset the count (and hence allow the user to retry logging in):

```
faillog -r -u user
```

where:

user is the name of the user account to unlock.

It is also possible to set the maximum number of password attempts on a per user basis, rather than system wide. If per-user limits are to be used, the PAM entries in the configuration file should be altered to:

```
auth      required     pam_tally.so onerr=fail deny=5 per_user
account   required     pam_tally.so reset
```

The `faillog` command is then used to set the per-user limit:

```
faillog -u user -m max
```

where:

max is the number of password failures before the account is disabled. A value of zero turns off checking.

user is the name of the user account for which to set the limit.

FreeBSD

Like Linux, FreeBSD also uses PAM to provide support for setting a maximum number of login attempts before a user is locked out. FreeBSD uses the `pam_abl` module to provide the required PAM support. The `pam_abl` package is not installed on a FreeBSD system by default. You may need to install the package before enabling it.

To enable the module you need to include it in the Texpress PAM configuration file located at `/etc/pam.d/texpress`. The following entry needs to be added:

```
auth    required    /usr/local/lib/pam_abl.so
config=/usr/local/etc/pam_abl.conf
```

The position of this line in the PAM configuration file is important. The line must appear **before** the entry for:

```
auth ... pam_unix.so
```

The `config` argument defines the location of the `pam_abl` configuration file. The configuration file allows system wide and per-user properties to be set.

To set a system wide entry, edit the configuration file located at `/usr/local/etc/pam_abl.conf` and alter the `host_rule` property. The format of the setting required to set the maximum login retries is:

```
*:retries/period
```

where:

retries is the number of login attempts before locking out the user.

period is the period in which the retries have to occur for the user to be locked out. The period is generally a number of days, for example `30d`. After the period has expired, the account may be accessed again.

To set a user specific entry, the `user_rule` should be used. The format of a user specific entry is:

```
user:retries/period
```

where:

user is the name of the user account to which the setting applies.

The remaining values are the same as for the `host_rule` property. More sophisticated rules may be set. Please see the manual entry for `pam_abl` for complete details.

The `pam_abl` command is used to clear an account once it has been disabled. The format of the command is:

```
pam_abl --okuser user
```

where:

user is the name of the user account to be re-enabled.

Windows

Disabling an account after a maximum number of login attempts is set on a Windows server running Vitalware via either a Local or Global Security Policy. The Local Security Policy editor is invoked by running `secpol.msc`. The Global Security Policy Editor is invoked by running `gpedit.msc`. The policy paths are:

- Local Security Policy
  `Security Settings/Account Policies/Account Lockout Policy`
- Global Security Policy
  [computer name] `Policy/Computer Configuration/Windows Settings/Security Settings/Account Policies/Account Lockout Policy`

The property used to control the maximum number of login attempts before disabling an account is:

- `Account lockout threshold` [the number of password attempts after which the account will be locked]

Once an account has been disabled, it must be re-enabled before the user can access Vitalware.

Examples

Example 1

Our institution has a policy of allowing five failed log in attempts before locking a user account. To unlock the account the user must contact the System Administrator.

Solaris 10

The setting required in the file `/etc/default/login` is:

```
RETRIES=5
```

and the setting required in the file `/etc/security/policy.conf` is:

```
LOCK_AFTER_RETRIES=YES
```

Linux

Add the following line to `/etc/pam.d/texpress`:

```
auth required pam_tally.so onerr=fail deny=5
```

The line should be added before the line:

```
auth ... pam_unix.so
```

FreeBSD

Install `pam_abl` and add the following line to the PAM configuration file located at `/etc/pam.d/texpress` (note that this should be entered on a single line):

```
auth required /usr/local/lib/pam_abl.so
config=/usr/local/etc/pam_abl.conf
```

The line should be added before the line:

```
auth ... pam_unix.so
```

Edit the `pam_abl` configuration file located at `/usr/local/etc/pam_abl.conf` and add the line:

```
host_rule=*:5/1h
```

The rule will allow up to five incorrect log in attempts per hour.

Windows

If the setting is to be domain wide, then the Global Security Policy should be updated; otherwise, if it is machine specific, the Local Security Policy should be used. The following attribute should be set:

`Account lockout threshold` set to `5`.

Example 2

We have one particular user (`boris`) who suffers from dyslexia, so we do not want to lock his account after five incorrect password submissions.

Solaris 10

The command required to disable account locking for user `boris` is:

```
usermod -K lock_after_retries=no boris
```

Linux

Edit the PAM configuration file located at `/etc/pam.d/texpress` and add the `per_user` property to the `pam_tally` auth entry:

```
auth required pam_tally.so onerr=fail deny=5 per_user
```

The command required to disable account locking for user `boris` is:

```
faillog -m 0 -u boris
```

FreeBSD

Edit the `pam_abl` configuration file located at `/usr/local/etc/pam_abl.conf` and add the following entry:

```
user_rule=boris:100/1s
```

While the above rule does not disable the checking, it will only lock out a user if there are more than 100 login attempts per second (which is very unlikely).

Windows

Windows does not provide a per-user setting for the number of login attempts before locking a user's account.

6. Valid Passwords

When a user submits a new password for updating it may be worthwhile checking that the password is sufficiently obscure as to not be easily guessed. The check may include looking for:

- known words
- a mixture of characters and digits
- a mixture of lower case and upper case characters
- a minimum length

The password validation facility provides a mechanism where new passwords can be checked against a number of criteria before being approved for updating.

Solaris 10

Solaris 10 provides support for checking that a new password supplied by a user meets a minimum set of conditions. The conditions are defined in the file `/etc/default/passwd` and are checked each time a user sets a new password. If the new password does not pass all checks, it is disallowed. The checks available are:

`DICTIONDBDIR`	The path of a directory containing a series of pre-compiled dictionary files. The files are checked to see if a new password appears in one of them. If so, the password is rejected. The `mkpwdict` command is used to build the pre-compiled dictionary files. The default value is empty, implying a dictionary check is not performed.
`DICTIONLIST`	A comma separated list of full file names to dictionary files. Each dictionary file should contain one word per line with a newline character used to end the line. A new password is checked against the contents of the dictionary files before being passed. The default value is an empty list, implying dictionary look ups are not performed.
`HISTORY`	The maximum number of prior passwords to maintain for each user. When a user submits a new password it is checked against the last `HISTORY` passwords the user has set and if a match occurs, the password is disallowed. If the value for `HISTORY` is empty or zero, password histories are disabled. The default value is empty, implying password histories are disabled. Password histories are only supported by the Shadow password database.
`MINALPHA`	The minimum number of alphabetic characters required. If `MINALPHA` is not set, the default is `2`.
`MINDIGIT`	The minimum number of digits required. If `MINDIGIT` is not set or is set to zero, the default is no checks. You cannot set `MINDIGIT` if `MINNONALPHA` is specified.

MINNONALPHA	The minimum number of non-alpha characters (including numeric and special) required. If MINNONALPHA is not set, the default is 1. You cannot set MINNONALPHA if either MINDIGIT or MINSPECIAL is specified.
MINSPECIAL	The minimum number of special characters (non-alphabetic and non-digit) required. If MINSPECIAL is not set or is zero, the default is no checks. You cannot set MINSPECIAL if you specify MINNONALPHA.
MINLOWER	The minimum number of lower case letters required. If not set or zero, the default is no checks.
MINUPPER	The minimum number of upper case letters required. If not set or zero, the default is no checks.
MINDIFF	The minimum character differences required between an old and a new password. If MINDIFF is not set, the default is 3.
NAMECHECK	Enable / disable checking for the user name as the password. The default is to perform the check. A value of NO disables this feature.
PASSLENGTH	The minimum acceptable length for a password in characters.
WHITESPACE	Whether whitespace characters (space, tab, etc.) are acceptable in a password. The default value is that whitespace is acceptable. A value of NO disables this feature.

As you can see Solaris 10 provides quite a bit of control over what constitutes a valid password. The settings above are on a system wide basis. It is not possible to apply any of the settings on a per-user basis.

Linux

Linux uses the pam_cracklib module in the PAM sub-system to implement password validation. In order to enable the module you need to include it in the Texpress PAM configuration file located at /etc/pam.d/texpress. The following two entries need to replace the existing password entry:

```
password     required      pam_cracklib.so minlen=6 difok=3
password     required      pam_unix.so use_authtok
```

The pam_cracklib module takes a number of parameters, including:

- minlen

  The minimum length for an acceptable password.

- difok

  The minimum number of characters difference between the previous password and the new password.

- dictpath

  Full file prefix for cracklib dictionaries. The default value is /usr/lib/cracklib.dict. The cracklib dictionary extensions are:

  - .hwm
  - .pwd
  - .pwi

pam_cracklib provides a dictionary containing over 50,000 words. It is also possible to configure pam_cracklib to enforce passwords to contain a mixture of lowercase, uppercase, digits and special characters. See the pam_cracklib manual page for complete details on how to configure these restrictions.

FreeBSD

FreeBSD uses the pam_passwdqc module in the PAM sub-system to implement password validation. In order to enable the module you need to include it in the Texpress PAM configuration file located at /etc/pam.d/texpress. The following two entries need to replace the existing password entry:

```
password    requisite     pam_passwdqc.so enforce=users
password    required      pam_unix.so no_warn try_first_pass
```

The pam_passwdqc module takes a number of parameters, including:

- enforce

  Indicates whether weak passwords should be allowed. The allowed values are:

  - everyone - strong password are enforced for all users.
  - users - strong passwords are enforced for all non-root users.
  - none - strong passwords are not enforced.

- max

  The maximum allowed password length. The default value is 40.

- match

The length of a common sequence between old and new passwords for the new password to be considered unsuitable. The default value is four, while a value of zero disables sub-string matching.

- `min`

  The minimum allowed password lengths for different combinations of character sequences. A password is broken down into a number of characters classes. The classes are:

  - digits
  - lower-case letters
  - upper-case letters
  - other characters

  The format of the setting for `min` is:

  `min=N0,N1,N2,N3,N4`

  where:

  `N0` is the minimum length where a password contains characters from one class only.

  `N1` is the minimum length where a password contains characters from two classes.

  `N2` is the minimum number of words for a passphrase (not used by Vitalware).

  `N3` is the minimum length where a password contains characters from three classes.

  `N4` is the minimum length where a password contains characters from all four classes.

  A value of disabled is used to disallow passwords of a given format. The default value is `min=disabled,24,12,8,7`.

`pam_passwdqc` does not provide a dictionary look-up facility, however judicious use of the `min` property forces users to submit passwords with non-obvious sequences.

Windows

Enabling password complexity requirements is set on a Windows server running Vitalware via either a Local or Global Security Policy. The Local Security Policy editor is invoked by running `secpol.msc`. The Global Security Policy Editor is invoked by running `gpedit.msc`. The policy paths are:

- Local Security Policy
  `Security Settings/Account Policies/Password Policy`
- Global Security Policy
  [computer name] `Policy/Computer Configuration/Windows Settings/Security Settings/Account Policies/Password Policy`

The property used to control password complexity is:

- *Password must meet complexity requirements* - specifies whether password complexity checks are enabled or disabled. The default value for a domain controller is enabled, otherwise disabled. The password complexity requirements are:
  - must not contain the user's account name or parts of the user's full name that exceed two consecutive characters.
  - must be at least six characters in length.
  - must contain characters from three of the following four categories:
    - upper-case characters (A through Z)
    - lowercase characters (a through z)
    - digits (0 through 9)
    - non-alphabetic characters (for example, !, $, #, %)

The property used to control password histories is:

- *Enforce password history* - specifies the number of unique new passwords required before an old password may be re-used. The default value is `24` for a domain controller, otherwise a value of zero is used.

Windows does not provide a dictionary look-up facility.

SECTION 4

PAM Configuration

PAM (Pluggable Authentication Modules) is a very flexible authentication system. As the name implies, it allows modules to be plugged in to provide specific functionality. Each module looks after some part of the authentication process with the combination of the results of each module determining whether access is granted.

For example, there is a module that provides LDAP functionality and another that provides Unix / Shadow functionality and so on. In order to provide the password checks and updates required by a given institution it is necessary to adjust the PAM configuration to match the institution's policy. If an institution uses Active Directory to manage users, then the PAM LDAP module must be enabled; if an institution uses dongles, then the required PAM module (e.g. `pam_usbng`) needs to be enabled.

It is beyond the scope of this document to explain how to configure PAM (there are plenty of good sources available on the internet). Rather we will look at configurations required to support the functionality required for password management on:

- Solaris 10
- Linux
- FreeBSD

In order to provide general support for the password database used by your institution (LDAP, AD, Shadow, etc.) within Vitalware you need to not only configure PAM, but also NSS (Name Service Switch). The combination of PAM and NSS on Unix systems provide the integration required to communicate with the various user / password databases. An explanation of NSS is beyond the scope of this document, however sample NSS configurations will be provided. The NSS configuration file is located at `/etc/nsswitch.conf`.

The PAM configurations outlined in this section apply to the setup required by Vitalware only. The configurations do not provide general purpose account access to the server via PAM, rather they allow Vitalware to be configured to use the required user / password database. The configurations show the settings required within the listed file, not the complete contents of the file. Thus if you are configuring the PAM and NSS settings you will need to amend the contents of the existing file, rather than replace them.

Solaris 10

The PAM configuration file used by Solaris 10 is located at `/etc/pam.conf`. The file contains the configuration for all PAM services, rather than one service per file (as is used by Linux and FreeBSD).

Shadow

The PAM and NSS configuration file segments required to provide *Shadow* database support are:

```
/etc/pam.conf
#
# Vitalware Texpress service
#
   texpress        auth sufficient         pam_rhosts_auth.so.1
   texpress        auth requisite          pam_authtok_get.so.1
   texpress        auth required           pam_dhkeys.so.1
   texpress        auth required           pam_unix_cred.so.1
   texpress        auth required           pam_unix_auth.so.1
/etc/nsswitch.conf
   passwd:     files
```

NIS & Shadow

The PAM and NSS configuration file segments required to provide NIS and Shadow database support are:

```
/etc/pam.conf
#
# Vitalware Texpress service
#
   texpress        auth sufficient         pam_rhosts_auth.so.1
   texpress        auth requisite          pam_authtok_get.so.1
   texpress        auth required           pam_dhkeys.so.1
   texpress        auth required           pam_unix_cred.so.1
   texpress        auth required           pam_unix_auth.so.1
/etc/nsswitch.conf
   passwd:     files nis
```

LDAP / AD & Shadow

The PAM and NSS configuration file segments required to provide LDAP or AD and Shadow database support are:

```
/etc/pam.conf
#
# Vitalware Texpress service
#
   texpress        auth sufficient         pam_rhosts_auth.so.1
   texpress        auth requisite          pam_authtok_get.so.1
   texpress        auth required           pam_dhkeys.so.1
   texpress        auth required           pam_unix_cred.so.1
   texpress        auth binding            pam_unix_auth.so.1
server_policy
   texpress        auth required           pam_ldap.so.1

   #
   # Default Account service
   #
   other           account requisite       pam_roles.so.1
   other           account binding         pam_unix_account.so.1
server_policy
   other           account required        pam_ldap.so.1

   #
   # Password checking (used by Admin Task only)
   #
   passwd          auth binding            pam_passwd_auth.so.1
server_policy
   passwd          auth required           pam_ldap.so.1

   #
   # Default Password service
   #
   other       password required           pam_dhkeys.so.1
   other       password requisite          pam_authtok_get.so.1
   other       password requisite          pam_authtok_check.so.1
   other       password required           pam_authtok_store.so.1
server_policy
/etc/nsswitch.conf
   passwd:     files ldap
```

If you select LDAP support, you will need to configure how to bind to the LDAP server. Use the `ldapclient` command to specify these settings.

Linux

The PAM configuration file used by Linux is located at `/etc/pam.d/texpress`. The file contains the configuration for Vitalware services only.

Shadow

The PAM file and NSS file segments required to provide Shadow database support are:

```
/etc/pam.d/texpress
  #
  # Vitalware Texpress service
  #
  auth        required      pam_env.so
  auth        required      pam_unix.so nullok try_first_pass

  account     required      pam_unix.so

  password    requisite     pam_cracklib.so try_first_pass
  password    required      pam_unix.so md5 shadow nullok
try_first_pass use_authtok
/etc/nsswitch.conf
  passwd:     files
  shadow:     files
```

NIS & Shadow

The PAM file and NSS file segments required to provide NIS and Shadow database support are:

```
/etc/pam.d/texpress
  #
  # Vitalware Texpress service
  #
  auth        required      pam_env.so
  auth        required      pam_unix.so nullok try_first_pass

  account     required      pam_unix.so

  password    requisite     pam_cracklib.so try_first_pass
  password    required      pam_unix.so md5 shadow nullok
try_first_pass use_authtok
/etc/nsswitch.conf
  passwd:     files nis
  shadow:     files nis
```

LDAP / AD & Shadow

The PAM file and NSS file segments required to provide LDAP or AD and Shadow database support are:

```
/etc/pam.conf
  #
  # Vitalware Texpress service
  #
  auth        required      pam_env.so
  auth        sufficient    pam_ldap.so
  auth        required      pam_unix.so nullok try_first_pass

  account     sufficient    pam_ldap.so
  account     required      pam_unix.so

  password    requisite     pam_cracklib.so try_first_pass
  password    sufficient    pam_ldap.so
  password    required      pam_unix.so md5 shadow nullok
try_first_pass use_authtok
/etc/nsswitch.conf
  passwd:     files ldap
  shadow:     files ldap
```

If you select LDAP support, you will need to configure how to bind to the LDAP server. The LDAP configuration file is located at `/etc/ldap.conf`. See the manual entry for `ldap.conf` for details on how to bind to an LDAP/AD server.

FreeBSD

The PAM configuration file used by FreeBSD is located at `/etc/pam.d/texpress`. The file contains the configuration for Vitalware services only.

Unix

The PAM file and NSS file segments required to provide Unix database support are:

```
/etc/pam.d/texpress
  #
  # Vitalware Texpress service
  #
  auth        required      pam_unix.so try_first_pass

  account     required      pam_login_access.so
  account     required      pam_unix.so

  password    requisite     pam_passwdqc.so enforce=users
  password    required      pam_unix.so try_first_pass
/etc/nsswitch.conf
  passwd:           files
  passwd_compat:    nis
```

NIS & Unix

The PAM file and NSS file segments required to provide NIS and Unix database support are:

```
/etc/pam.d/texpress
  #
  # Vitalware Texpress service
  #
  auth        required      pam_unix.so try_first_pass

  account     required      pam_login_access.so
  account     required      pam_unix.so

  password    requisite     pam_passwdqc.so enforce=users
  password    required      pam_unix.so try_first_pass
/etc/nsswitch.conf
  passwd:           files nis
  passwd_compat:    nis
```

LDAP / AD & Unix

The PAM file and NSS file segments required to provide LDAP or AD and Unix database support are:

```
/etc/pam.conf
  #
  # Vitalware Texpress service
  #
  auth        sufficient    /usr/local/lib/pam_ldap.so
try_first_pass
  auth        required      pam_unix.so try_first_pass

  account     required      pam_login_access.so
  account     sufficient    /usr/local/lib/pam_ldap.so
  account     required      pam_unix.so

  password    requisite     pam_passwdqc.so enforce=users
  password    sufficient    /usr/local/lib/pam_ldap.so use_authtok
  password    required      pam_unix.so try_first_pass
/etc/nsswitch.conf
  passwd:           files ldap
```

If you select LDAP support, you will need to configure how to bind to the LDAP server. The PAM LDAP configuration file is located at:

`/usr/local/etc/ldap.conf`

A copy of the configuration file should be linked to:

`/usr/local/etc/nss_ldap.conf`

to provide the required NSS LDAP configuration.

Index

Vitalware®
Vital Records Management

The World's Premier Vital Records Software

Vitalware Documentation

Dynamic Security

Document Version 1.2

Vitalware Version 2.0

KE Software

www.kesoftware.com

Contents

SECTION 1

Dynamic Security

The default security model used by Vitalware is static in nature. User and Group privileges are defined in the Vitalware Registry and loaded into a module when it is invoked. Once invoked the security of the module remains the same throughout its lifetime. If a user can change the contents of a given field, then they can change it for all records (assuming Record Level Security allows the record to be modified). In some instances it would be useful to allow some security settings to be altered based on information stored in the current record.

For example:

1. The Data Entry group are able to change all fields on unregistered records and only non-certificate fields on registered records. The current Vitalware security model does not provide a mechanism for implementing this requirement via Registry settings. It is possible to "hard wire" such functionality into the Vitalware client, however it becomes very difficult to change as new requirements arise.
   What would be useful is a mechanism that allows access to a column to be modified based on the contents of the record.
2. Similarly, you may require certain fields to be filled depending on the type of record. For Registration records you may require the *Informant* information to be specified, while for Index records this information should not be specified (in fact, it should not even be shown). The Vitalware Mandatory Registry entry allows a field to be defined as mandatory, however it is not possible to use this Registry entry to specify conditional mandatory settings.
   As with the first example, it would be useful to allow the mandatory setting for a field to be set based on the contents of the record.
3. Alternatively, you may want to alter Record Level Security settings based on the contents of data within the record when the record is saved. One such requirement may be that records whose record status is set to `Voided` may only be edited by users in group Admin. Such a feature can be "hard wired" into the database server.

However a solution that uses the Registry would provide a more flexible mechanism.

With the addition of two Registry settings, Vitalware 2.2.02 implements a flexible and dynamic security module which can adapt based on the data stored within a record:

- The Column Access Modifier Registry entry handles the first example above, that is the ability to alter column access based on the contents of the record.
- The Mandatory Modifier Registry entry handles the second example above, that is the ability to adjust mandatory settings based on the contents of the record.
- The Security Update Registry entry provides for the second example, that is the ability to change Record Level Security based on the contents of the current record.

In this document we look at all three Registry entries and explain how they may be used to provide a dynamic security model, i.e. one that changes based on the data in the current record.

SECTION 2

Security Update

The Security Update Registry entry allows the contents of one or more fields to be modified based on the value in a field whenever a record is saved (inserted or modified). Importantly, the Record Level Security (RLS) fields:

- `SecCanDisplay`
- `SecCanEdit`
- `SecCanDelete`

may be adjusted, allowing the record security to be modified. However, the Registry entry is not restricted to RLS fields, and any combination of fields may be adjusted.

The format of the entry is:

`User|`*user*`|Table|`*table*`|Security|Update|`*column*`|`*value*`|`*settings*

`User|`*user*`|Table|Default|Security|Update|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|`*table*`|Security|Update|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|Default|Security|Update|`*column*`|`*value*`|`*settings*

`Group|Default|Table|`*table*`|Security|Update|`*column*`|`*value*`|`*settings*

`Group|Default|Table|Default|Security|Update|`*column*`|`*value*`|`*settings*

where:

column defines which field should be consulted to look for a matching *value*.

value is a Vitalware based search pattern.

If there is a match of the data in *column* with the *value* query, then the Registry entry is used and *settings* is applied.

If *column* contains a table of values, each entry is checked against *value*.

Since *value* is a pattern, particular attention must be paid if you want to match the complete contents of a field.

For example, to match `Pending` but not `Pending Payment`, the pattern `^Pending$` should be used. It is important to remember that *value* operates in the same way as an Vitalware search term.

All comparisons of *value* are case insensitive.

settings is a semi-colon separated list of assignments to columns that is applied if there is a match of the data in *column* with the *value* query.

The format of *settings* is:

column`=[+/-]`*term*`:[+/-]`*term*`:...;`*column*`=[+/-]`*term*`:...`

where:

column is the name of the column to be modified.

term is the value to be set in *column.*

- If *term* is preceded by a plus symbol (+), the value is added to the existing list of values.
- If *term* is preceded by a minus sign (-), the value is removed from the existing list of values.
- If no symbol precedes a *term*, the current contents of *column* are removed and replaced with *term*.
- If more than one *term* is supplied for a given column, separated by colons (:), each *term* is applied one after the other.
- If more than one column is to be modified, each set of column settings is separated by a semi-colon (;).

Examples

If we want Record Level Security to be adjusted so that users in group Adoptions are the only ones allowed to view, edit and delete records that have a record status of `Adoption`, the following entry can be used:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`Default`
Key 5	`Security`
Key 6	`Update`
Key 7	`RecordStatusFlag`
Key 8	`^A$`
Value	`SecCanView=Group Adoption;SecCanEdit=Group Adoption;SecCanDelete=Group Adoption`

Keys 7 and 8 indicate that the entry only applies where the *RecordStatusFlag* field matches the pattern `^A$` (i.e. where the field contains `A` only). If this is not the case, then the Registry entry is ignored. Where a record does match, the *SecCanEdit* field is set to `Group Adoption`. As a leading plus or minus is not supplied, the contents of *SecCanEdit* are replaced with `Group Adoption`. A similar update occurs for *SecCanDelete* and *SecCanView*.

A more complex example would involve removing edit access for all users in group Data Entry when a record is cancelled. Let's assume that the field *RecOrdStatus* contains the word `Cancelled` for records that are no longer valid registrations. A suitable Registry entry would be:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ebirths`
Key 5	`Security`
Key 6	`Update`
Key 7	`RecOrdStatus`
Key 8	`^Cancelled$`
Value	`SecCanEdit=-Group Data Entry`

Notice how the terms to set have a leading minus sign, indicating the term (in this case the group name) is to be removed from the field *SecCanEdit*.

A third example requires all records with a print status of `Ready` to have group Counter removed and group Print added for both displaying and editing the record. The field containing the print status is *ReadyToPrint*. A suitable Registry entry is:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`epos`
Key 5	`Security`
Key 6	`Update`
Key 7	`ReadyToPrint`
Key 8	`^Y$`
Value	`SecCanDisplay=-Group Counter:+Group Print; SecCanEdit=-Group Counter:+Group Print`

Notice how more than one field may be updated with a single Registry entry. Note also:

- If a term has a leading plus symbol and the term already appears in the field, it is not added again.
- Similarly, if a term has a preceding minus and it does not appear in the field, it is ignored.

In this final example we restrict the display privilege for records that have not been approved for viewing on the intranet to groups Admin, Registrations and Adoptions. Once the record has been approved for viewing on the intranet we will allow all users to view the record. In this case two Registry entries are required:

- The first restricts access for records not available on the intranet.
- The second allows access to all users for record available on the intranet.

Suitable Registry entries are:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ebirths`
Key 5	`Security`
Key 6	`Update`
Key 7	`AdmPublishWebPasswordFlag`
Key 8	`N`
Value	`SecCanDisplay=Group Admin:+Group Registrations:+Group Adoptions`

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ebirths`
Key 5	`Security`
Key 6	`Update`
Key 7	`AdmPublishWebPasswordFlag`
Key 8	`Y`
Value	`SecCanDisplay=Group Default`

Notice how the first term in the first Registry entry (`Group Admin`) does not have a leading plus or minus, meaning it replaces the current contents of *SecCanDisplay*. The following terms require a leading plus symbol otherwise they will also clear the current contents rather than adding to the first term. The second Registry entry replaces the contents of *SecCanDisplay* with group Default (this allows access for everyone).

By using a combination of Registry entries it is possible to produce quite sophisticated and dynamic privilege changes.

What's happening behind the scenes

The Security Update Registry entry is a Record Level Security based Registry entry and like all RLS Registry entries it is enforced by the database engine. The `security` file stored in the table directory provides the configuration used for RLS. The XML format of the `security` file has been extended to allow the contents of the Security Update Registry entry to be accommodated. Consider this entry:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ebirths`
Key 5	`Security`
Key 6	`Update`
Key 7	`RecordStatus`
Key 8	`^Cancelled$`
Value	`SecCanEdit=Group Admin:+Group Registration;SecCanDelete=Group Admin:+Group Registration`

The extra XML generated in the `security` file for this entry is:

```
<updates>
  <update name="RecordStatus" value="^Cancelled$">
    <columns>
      <column name="SecCanEdit">
        <values>
          <value operation="replace" term="Group Admin"/>
          <value operation="add" term="Group Registration"/>
        </values>
      </column>
      <column name="SecCanDelete">
        <values>
          <value operation="replace" term="Group Admin"/>
          <value operation="add" term="Group Registration"/>
        </values>
      </column>
    </columns>
  </update>
</updates>
```

As you can see the XML follows the sequencing of the Registry entry. If multiple Registry entries exist, the `<update>` tags are repeated.

The good news is that you do not need to add the XML to the `security` file. Whenever a security based Registry is added or modified in Vitalware, the `security` file is rebuilt automatically, and all you need to do is add the Registry entries.

The order of processing

The database server applies the Security Update settings whenever a record is saved (i.e. for all insertions and updates). The entries are applied after assignment expressions have been executed and **before** validation is run. This means that assignment expressions may be used to build a composite value that may be tested by Security Update settings. For example, it is possible to concatenate two fields into one that may then be checked.

It is also possible to apply sophisticated formula to calculate a field to be checked. For example, you may want to set Security Update Registry entries based on a range of years for a registration. You could use an assignment expression to set a value in a field based on the ranges:

Range	Value
<1913	`Public`
1913-Today	`Embargoed`

You may then use these values, `Public` and `Embargoed` in Security Update Registry entries. Note that you cannot use validation code to compute values as the Security Update entries are applied before validation code is executed.

SECTION 3

Column Access Modifier

As the name suggests, the Column Access Modifier Registry entry is used to modify the default Column Access values based on data found in the current record.

The format of the Registry entry is:

`User|`*user*`|Table|`*table*`|Column Access Modifier|`*column*`|`*value*`|`*settings*

`User|`*user*`|Table|Default|Column Access Modifier|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|`*table*`|Column Access Modifier|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|Default|Column Access Modifier|`*column*`|`*value*`|`*settings*

`Group|Default|Table|`*table*`|Column Access Modifier|`*column*`|`*value*`|`*settings*

`Group|Default|Table|Default|Column Access Modifier|`*column*`|`*value*`|`*settings*

where:

column defines which field should be consulted to look for a matching *value*.

value is a case-insensitive match (i.e. patterns/wildcards are not allowed unlike with the Security Update entry (page 3)). The *value* is checked against the complete contents of *column*. The comparison is case insensitive. If *column* is a table of values, then each entry in the table is tested. If one matches, the Registry entry applies. Values of `NULL` and `NOT NULL` may be used to represent empty and non-empty values respectively.

settings is a semi-colon separated list of assignments to columns that is applied if there is a match of the data in *column* with the *value* query.

The format of *settings* is:

column`=[+/-]`*perm*`:[+/-]`*perm*`:...;`*column*`=[+/-]`*perm*`:...`

where:

column is the name of the column to be modified.

perm is the permission to be adjusted.

- A leading plus sign is used to add a permission.
- A leading minus sign removes a permission.
- If no sign is supplied, the current permissions are replaced (this works the same as for the Security Update entry).

The list of possible values for *perm* is:

- `dvDisplay` - see column while viewing a record.
- `dvEdit` - see column while modifying a record.
- `dvInsert` - see column while inserting a record.
- `dvQuery` - see column while searching for a record.
- `duEdit` - change column while modifying a record.
- `duInsert` - change column while inserting a record.
- `duQuery` - change column while searching for a record.
- `duReplace` - use column in a global replace command.

Examples

For this example, we want to ensure the *Stock Number* field cannot be updated for issued certificates, i.e. we want to remove the `duEdit` permission where the *Verified* field is `Y`. A suitable entry is:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecertificates`
Key 5	`Column Access Modifier`
Key 6	`StoVerified`
Key 7	`Y`
Value	`StoStockNumberText=-duEdit`

Now when certificate records are displayed, the *Stock Number* field will be greyed out (indicating that it cannot be modified). To restrict the above entry to users in group Print only, you would need to change Key 2 to `Print`.

The Column Access Modifier Registry entry is generally set on a group or user basis.

Unlike the Security Update Registry entry, where changes are applied when a record is saved, Column Access Modifier entries are applied immediately. If the content of a column is changed and it matches an entry, the entry is applied at once. Also, after applying any Column Access Modifier entries, any column affected by previous changes but not updated with the current changes will be reset to its default setting (as defined by the Column Access Registry setting).

In this next example, we want group Call Centre to be able to change the *Notes* field for POS records that are not cancelled. In order to provide this setting the default Column Access settings must have `duEdit` enabled for group Call Centre. The Registry entries required are:

Field	Value
Key 1	`Group`
Key 2	`Call Centre`
Key 3	`Table`
Key 4	`epos`
Key 5	`Column Access`
Key 6	`NotNotes`
Value	`dvQuery;dvDisplay;dvEdit;duEdit;duQuery`

Field	Value
Key 1	`Group`
Key 2	`Call Centre`
Key 3	`Table`
Key 4	`epos`
Key 5	`Column Access Modifier`
Key 6	`RecOrdStatus`
Key 7	`Cancelled`
Value	`NotNotes=-duEdit`

The first entry sets the default permissions for the column *NotNotes*. Notice that `duEdit` is enabled by default. This allows users in group Call Centre to change the contents of the field.

The second entry modifies the default settings to turn off `duEdit` where the record is cancelled.

The above entries show how the Column Access Modifier Registry entry may be used with the Column Access Registry entry to provide a predicable set of permissions for all record status values.

In this last example we disable editing for all users in group Registrations of the *AKA Name* field until a value is entered into the *Surname* field. The restriction is to apply when creating records as well as modifying records. The following entry could be used:

Field	Value
Key 1	`Group`
Key 2	`Registrations`
Key 3	`Table`
Key 4	`edeaths`
Key 5	`Column Access Modifier`
Key 6	`DeceasedSurname`
Key 7	`NULL`
Value	`DeceasedAlternateName=-duEdit:-duInsert`

You may have been tempted to use the following entries:

Field	Value
Key 1	`Group`
Key 2	`Registrations`
Key 3	`Table`
Key 4	`edeaths`
Key 5	`Column Access Modifier`
Key 6	`DeceasedSurname`
Key 7	`NULL`
Value	`DeceasedAlternateName=-duEdit:-duInsert`

Field	Value
Key 1	`Group`
Key 2	`Registrations`
Key 3	`Table`
Key 4	`edeaths`
Key 5	`Column Access Modifier`
Key 6	`DeceasedSurname`
Key 7	`NOT NULL`
Value	`DeceasedAlternateName=+duEdit:+duInsert`

to ensure that users can edit the *AKA Name* field if the surname is filled. The second Registry entry forces the edit and insert privileges to be enabled even if the default Column Access settings do not allow it. It may be your intention to have this behaviour, however the original brief only stipulated that edit and insert permissions should be disabled for *AKA Name* if the surname is empty. It is not expressed what the permissions should be if the main title is filled. Another way of writing the second set of Registry entries is:

Field	Value
Key 1	`Group`
Key 2	`Registrations`
Key 3	`Table`
Key 4	`edeaths`
Key 5	`Column Access`
Key 6	`DeceasedAlternateName`
Value	`dvQuery;dvDisplay;dvEdit;dvInsert;duEdit; duInsert;duQuery;duReplace`

Field	Value
Key 1	`Group`
Key 2	`Registrations`
Key 3	`Table`
Key 4	`edeaths`
Key 5	`Column Access Modifier`
Key 6	`DeceasedSurname`
Key 7	`NULL`
Value	`DeceasedAlternateName=-duEdit:-duInsert`

In this case the default column permissions enable edit and insert privileges, while the second Registry entry disables edit and insert if the main title is empty. If the main title is filled, the default permissions are applied, hence enabling edit and insert privileges.

SECTION 4

Mandatory Modifier

The Mandatory Modifier Registry entry is used to modify the mandatory setting for a given field based on data found in the current record.

The format of the Registry entry is:

`User|`*user*`|Table|`*table*`|Mandatory Modifier|`*column*`|`*value*`|`*settings*

`User|`*user*`|Table|Default|Mandatory Modifier|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|`*table*`|Mandatory Modifier|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|Default|Mandatory Modifier|`*column*`|`*value*`|`*settings*

`Group|Default|Table|`*table*`|Mandatory Modifier|`*column*`|`*value*`|`*settings*

`Group|Default|Table|Default|Mandatory Modifier|`*column*`|`*value*`|`*settings*

where:

column defines which field should be consulted to look for a matching *value*.

value is a case-insensitive match (i.e. patterns/wildcards are not allowed unlike with the Security|Update entry). The *value* is checked against the complete contents of *column*. The comparison is case insensitive. If *column* is a table of values, then each entry in the table is tested. If one matches, the Registry entry applies. Values of `NULL` and `NOT NULL` may be used to represent empty and non-empty values respectively.

settings is a semi-colon separated list of assignments to columns that is applied if there is a match of the data in *column* with the *value* query.

The format of *settings* is:

column=*setting*;*column*= *setting*;. . .

where:

column is the name of the column to be modified.

setting is `true` (the column should be mandatory) or `false` (the column should not be mandatory, i.e. remove the mandatory setting).

Examples

Example 1

For this example, we want the *Informant Name* field to be mandatory, but only if the registration type is `Full`. A suitable entry is:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ebirths`
Key 5	`Mandatory Modifier`
Key 6	`RegistrationType`
Key 7	`Full`
Value	`InformantName=true`

Now when the registration type is set to `Full`, the *Informant Name* field must be filled before the record can be saved successfully. If *Informant Name* is not completed, the standard error message is displayed. You may tailor the error message shown using the Mandatory Registry entry. For example:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ebirths`
Key 5	`Mandatory`
Key 6	`InformantName`
Value	`False;Please enter an Informant Name for this Birth`

will display the error message *Please enter an Informant Name for this Birth* if the field is not filled while the mandatory setting is `true`. A `false` value indicates the field is not mandatory by default.

Example 2

The above example shows how to alter the mandatory setting for a field (*InformantName*) based on the value in another field (*RegistrationType*). In this example the mandatory setting for more than one field is altered based on the values of multiple fields. If the registration type is set to `Full` and the *AdoptionFlag* is set to `Yes`, then the *DateOfAdoptionDecrees* and *DecreeAuthority* fields must be supplied:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ebirths`
Key 5	`Mandatory Modifier`
Key 6	`RegistrationType`
Key 7	`Full`
Value	`DateOfAdoptionDecree=true;DecreeAuthority=true;TitAccessionLot=true`

Field	Value
Field	`Value`
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ebirths`
Key 5	`Mandatory Modifier`
Key 6	`AdoptionFlag`
Key 7	`Yes`
Value	`DateOfAdoptionDecree=true;DecreeAuthority=true;TitAccessionLot=true`

Where multiple values are to be checked, a Mandatory Modifier Registry entry is required for each value for each column. Where multiple Registry entries match based on the values within fields, the mandatory settings are ANDed together. This means that unless all the settings for a given field are `true`, mandatory is set to `false`.

SECTION 5

Conclusion

Vitalware 2.0 sees the addition of three new Registry entries. The first, Security|Update, allows record level permissions to be altered when a record is saved, based on the contents of the record. The second, Column Access Modifier, allows field based privileges to be altered based on the contents of the record. The third, Mandatory Modifier, allows mandatory fields to be specified based on the contents of the record. The combination of the three facilities provides a useful mechanism for altering the Vitalware security settings dynamically. The use of dynamic security allows very flexible security models to be implemented.

Index

Vitalware®
Vital Records Management

The World's Premier Vital Records Software

Vitalware Documentation

Lookup List Maintenance

Document Version 1

Vitalware Version 2.3.01

KE Software

www.kesoftware.com

Contents

SECTION 1

Overview

Lookup Lists have been a part of Vitalware since the first version of the software. They provide a useful mechanism for terminology control and are used extensively in the Vitalware client. The Lookup List facility stores data in a database table called eluts. The table contains a single record for each unique Lookup List entry. The information recorded for an entry includes:

`Name` The name of the Lookup List. The name is used to associate a set of Lookup List entries with a particular field in the Windows client.

`Values` A Lookup List entry may contain a number of values, with each value being a level in a hierarchy. The values start at level zero and increase. If a Lookup List is not part of a hierarchy (that is, it does not contain multiple levels), then only value zero is set. For hierarchies, a record will exist for each level in the hierarchy.

For example, if we have a Lookup List called Location, consisting of three levels:

- Country
- State
- City

with the data:

- `Australia` (Country)
- `Victoria` (State)
- `Bendigo` (City)

then three Lookup List entries (records) are generated:

	Entry 1	**Entry 2**	**Entry 3**
Country	`Australia`	`Australia`	`Australia`
State		`Victoria`	`Victoria`
City			`Bendigo`

The reason for three entries is that if a user wants to view a list of all countries (by viewing the *Countries* Lookup List for instance), then all Location Lookup entries that have the first value only filled are retrieved. As users may ask for any level in the hierarchy, the three records satisfy any potential request.

`Levels` The number of levels filled for the current record.

Using the example above, *Entry 1* has a `Levels` value of 1 (as only a single level is filled), while *Entry 2* has a `Levels` value of 2, etc.

`Persistent` If a Lookup List entry is not used anywhere in the system, the entry is removed from the Lookup List table. For example, if a record does not exist with a city location of `Bendigo`, then Entry 3 in the table above would be removed.

A `Persistent` Lookup List entry is not deleted if the entry is not used.

`Persistent` entries are used to pre-populate Lookup Lists with a known set of values, even if the values are not currently in use.

`Hidden` If a Lookup List entry is marked as `Hidden`, then the entry is shown for searches but not when inserting new values.

For example, if the city `Bendigo` changed its name to `Sandhurst`, a new Lookup List entry is created with the city value `Sandhurt`. As the data still contains the value `Bendigo`, the Lookup List entry for this value is not removed. When users are inserting new records, we do not want the value `Bendigo` to appear in the Lookup List (as `Sandhurst` is the correct entry). However, when performing a search we still want `Bendigo` to appear in the list as there are still records containing this value. The `Hidden` attribute provides this functionality.

The `Hidden` setting is used to phase out entries that should no longer be used.

`Used` The `Used` flag indicates whether the Lookup List entry is used anywhere within the system. If the flag is enabled, then at least one record in Vitalware uses the Lookup List entry's value.

`SortOrder` By default, the entries in a Lookup List are displayed in alphabetic order. It is possible to display the entries in a user specified (i.e. customised) order. The `SortOrder` value is used as the sort key for sorting the values in a Lookup List with customised ordering. The value may be numeric or alphabetic, in which case a numeric or alphabetic sort is applied respectively.

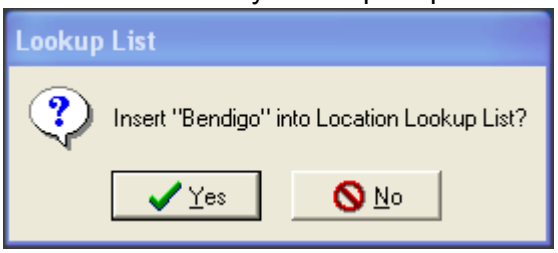
The `SortOrder` attribute is rarely used.

In order to implement customised sorting for a given Lookup List, the Windows client must be configured to provide the required functionality. Please contact KE Support for details.

"Dirty" Lookup Lists

There are a number of issues with the storage of Lookup List entries in the eluts database table:

1. Ideally, when deleting a record, any values in the deleted record which are in a Lookup List should be checked for uniqueness: if the values are not used in any other records, then the corresponding entry in the eluts database table should be deleted. However, the time required to perform these checks is prohibitive and in the interests of efficiency Vitalware does not perform them when a record is deleted. As a result Lookup List entries may exist in the eluts table where the value is not used in any records in the system.
2. If a user edits a Lookup value in a record and replaces it with a new value, when the record is saved they will be prompted to add the new value to the eluts table:



However, the old value may not be used in other records in the system. As with the first issue, the time required to perform the check would dramatically increase the time required to save a record and, again, Vitalware does not perform the checks in the name of efficiency.

A consequence of these issues is that the eluts table can become "dirty", containing entries that are no longer required and which users should not be seeing.

To solve the problem of "dirty" Lookup Lists, Vitalware rebuilds the contents of the eluts table on a nightly basis (or as defined by the system maintenance schedule). The rebuild process may be quite time consuming for sites with large numbers of records. Since the rebuild process reads the Lookup List values for all records in Vitalware, it can build a new version of the eluts table containing only the correct entries. Once the rebuild is complete, the eluts table is back in sync with the Vitalware data.

The need to rebuild the Lookup List table each night means that the Lookup Lists are offline while the rebuild takes place. It also means that there is less time to perform other nightly maintenance routines (e.g. batch updates, etc.). Another issue is that since the contents of the eluts table are replaced each night, users cannot use records in the eluts table in the way they can for any other module. For a given entry it is not possible to:

- Add notes or multimedia.
- Follow audit trails on changes made.
- Set record level security.

It is also not possible to add new values to a Lookup List by simply adding a new record to the eluts table.

New Lookup Lists background service

To reduce the nightly system maintenance and to allow the Lookup List table to be used as a regular module, Vitalware 2.3.01 has added a background service that ensures the Lookup List entries are always in sync with the records in the Vitalware system. The addition of the service removes the need to rebuild the Lookup Lists on a nightly basis, hence reducing system maintenance time.

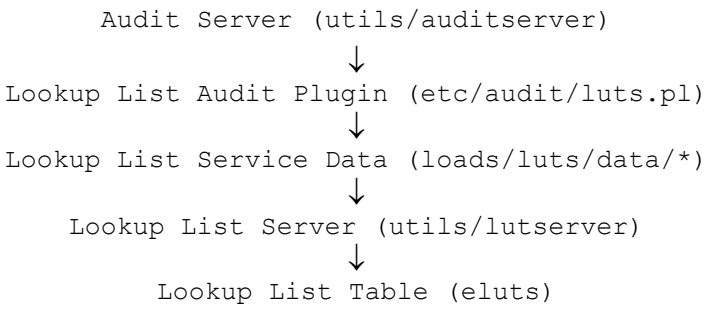
Furthermore, as the Lookup List table is no longer reloaded, the Lookup List module has been extended to include support for attributed notes (Notes tab) and multimedia (Multimedia tab). Audit trails on individual records are now maintained and Record Level Security may be used as for any other module.

Although a Lookup List rebuild program is still provided, it now applies changes to the eluts table rather than rebuilding the table completely. The introduction of the Lookup List service means that rebuilds are only required if the Lookup List table becomes corrupted, or if users accidentally delete records that are still in use.

lutserver

The server that handles updating of the eluts table in the new Lookup List service is called lutserver. lutserver runs on the Vitalware server machine waiting for requests to update Lookup List entries. The requests are generated by an audit trail plugin that picks up all deleted and modified records and determines what Lookup Lists need to be updated.

The process may be shown diagrammatically as:



Let's use an example to describe the steps taken by the Lookup List service to ensure Lookup Lists are kept up to date. In this example a user changes the city value in a record from `Bendigo` to `Sandhurst` and saves the record. The following steps occur:

1. When the user saves the record, the Vitalware server generates an XML description of the changes made to the record. The Audit Server (auditserver) loads these changes and passes them on to all registered plugins. The Lookup List Services registers the Lookup List Audit Plugin when the Audit Server is started. For our example record, the changes will show that `Bendigo` was changed to `Sandhurt` in the *City* column.
2. The Lookup List Audit Plugin looks at the changes supplied by the Audit Server. For each column changed it checks to see if a Lookup List is associated with the column. For our example, it will determine that the *City* column is associated with the *Location* Lookup List. It will then look at the two values and their associated operations. The value `Bendigo` has a delete operation (since it is being removed) and the value `Sandhurt` has an insert operation (as it is the new value). Since the Windows client was used to save the record, the Audit Plugin will ignore the insert operation as the user will have been asked to add the new entry to the *Location* Lookup List if it did not exist already. The delete operation is passed to the Lookup List Server for processing.
3. The Lookup List Audit Plugin writes a file containing the:
   - Operation performed (delete)
   - Value deleted (`Bendigo`)
   - Lookup List Name (*Location*)
   - Column changed (*City*)

   The file is located in the `loads/luts/data` directory. The format of the file name is `date.time.table.irn`, where:
   - `date` is an eight digit date in `yyyymmdd` format.
   - `time` is a six digit time in `hhmmss` format.
   - `table` is the database table in which the record was modified.
   - `irn` is the key number of the record modified.

   An example file name is: `20121026.133308.epos.375`.

4. The Lookup List Server fetches all files in the `loads/luts/data` directory and sorts them into date/time order. It is important that the files are processed in the same order as they were created, otherwise synchronisation issues may arise. The Lookup List Server reads the contents of each file, extracting the information within. Then for each Lookup List in the file it determines the operation to apply (insert or delete):
   - For an insert operation it checks to see if the value is already in the Lookup List; if not it inserts a new value into the eluts table.
   - For a delete operation it checks whether the value is used anywhere in the Vitalware system for the given Lookup List name. If the value is not used, the record is deleted from the eluts table (provided it is not `Persistent`).

   Once the file has been processed, it is removed. Once all the files have been processed, the Lookup List Server waits for new files to process.

The reason for so many steps is that audit plugins must be fast so that the Audit Server can process audit records quickly. To ensure the audit plugin is fast, the main Lookup List processing is removed from the audit plugin and moved to the Lookup List Server. The split ensures that the Audit Server keeps up with audit changes, even when the Lookup List Server may lag.

The Lookup List Server handles all deletion operations, that is it checks whether a Lookup List entry is still in use and if not, deletes it. It also handles insertions (that is new values) from all sources except the Windows client.

The Lookup List Server and the Registry

The Lookup List Server complies with the Lookup and Lookup Exact Registry entries:

- If a column has Lookup Exact set to `true`, then all comparisons with existing entries in the Lookup List table are performed as exact matches, i.e. the character case and punctuation must match exactly.
- If Lookup Exact is not enabled, all punctuation and character case is ignored for value comparisons. Search the Vitalware Help for details on the `Lookup Exact Registry entry` for more details.

The Lookup Registry entry is used to control the flags set when adding new values to a Lookup List. The table below lists each setting and examines how the Lookup List Server implements the required functionality. The Lookup Registry entry settings are generally set on a per column basis.

Setting	Description
`skip` `readonly`	The entry will not be added to the Lookup List table. If the column is part of a hierarchy, the entry is only skipped if the bottom level has this setting enabled. For example, if the *State* column has skip enabled, then entries with *Country*, *State* and *City* will be created, but entries with just *Country* and *State* will be skipped.
`readwrite`	The `readwrite` setting adds a new value to the Lookup List table. If an entry already exists but has the `Hidden` flag enabled, the flag will be reset (i.e. disabled). If the `Used` flag is disabled, it is enabled.
`autowrite`	The `autowrite` setting adds a new value to the Lookup List table. If an entry already exists and the `Used` flag is disabled, it is enabled.

`autowriteignore`
`readignore`
`writeignore`
	If the entry does not exist, a new entry is created with `Hidden` enabled. If an entry already exists and the `Used` flag is disabled, it is enabled.

Finally, the Lookup List Server will not delete any entry from the Lookup List table that has `Persistent` enabled.

The Lookup List Server, lutserver, provides the functionality required to keep the Lookup List table in sync with values used within the Vitalware System. The addition of the server removes the need to rebuild the Lookup List tables on a nightly (or otherwise) basis.

vwlutsrebuild

Prior to Vitalware 2.3.01, `vwlutsrebuild` was used to rebuild the Lookup List tables on a nightly basis. Functionality added with Vitalware 2.3.01 means that Lookup List tables no longer need to be rebuilt and although in theory `vwlutsrebuild` is no longer required, there may be occasions when the Lookup List tables need to be updated manually. It would be useful for example to be able to regenerate a missing entry if someone deleted a record accidentally.

The `vwlutsrebuild` server side program has been rewritten to work with the Lookup List Server framework (introduced with Vitalware 2.3.01). The program no longer generates a data file with all Lookup List entries in it that is then loaded into an empty Lookup List table. Instead, the program now visits each Vitalware record and checks that the values in the records are in the Lookup List table. If a value is not present, a new entry is created. Any entries that are no longer used, are deleted unless they are `Persistent`.

The rebuild process occurs in two stages. The first stage involves generating a list of all changes that need to be applied to the Lookup List table. This stage is known as the data generation phase. There are three types of changes possible:

- *insert* - new records to be added to the Lookup List table.
- *update* - existing records that require their flags (`Hidden`, `Used`, `Persistent`) to be adjusted.
- *delete* - existing records that are no longer required.

The data for each of the above changes is placed in the `luts` directory in files named `data.insert`, `data.update` and `data.delete` respectively.

Once the data generation phase is complete, the loading phase is executed. In this phase the files generated are loaded into the Lookup List table. Once the load is finished, the Lookup List table rebuild is complete.

The usage message for `vwlutsrebuild` is:

```
Usage: vwlutsrebuild [-dlmqtv] [lookup list ...]
-d    only produce lookup table data loading files
-f    load lookup changes quickly (takes eluts offline)
-l    only load lookup table data file
-t    include lookup list text files in rebuild
-v    verbose mode, print debugging information
```

The following table describes each option:

Option	Description
`-d`	Run the data generation phase only: the data files to be loaded are generated but not applied.
`-l`	Skip the data generation phase and run the loading phase only. The data files to be loaded are assumed to exist.
`-f`	Determine whether the Lookup List table should be taken offline while the loading phase is underway. If the `-f` option is specified, the `eluts` table will be taken offline and the data loaded, otherwise the table will be left online while the load proceeds. The load will be much faster for large amounts of data if the `-f` option is specified.
`-v`	Useful for tracing what the data generation phase is producing. Each entry to be inserted, updated or deleted is printed to the screen (as well as added to the appropriate data files).

`-t`	Informs `vwlutsrebuild` to include text files found in the `luts/defaults` and the `local/luts/defaults` directories as part of the data generation phase. The text files are used to load pre-built Lookup List values. Entries found in these files are loaded with `Persistent` enabled. The format of the text file is:
	`#`
	`# Lines beginning with a hash character are comments`
	`#`
	`[-]`*lookup list name*`|`*value 1*`[|`*value 2*`|...][=`*sortorder*`]`
	where:
	lookup list name — is the name of the Lookup List into which the value is to be loaded.
	Value 1 — is the value for the first level of the Lookup List. For hierarchies, the successive levels are separated by a pipe (|) symbol.
	sortorder — An optional equals sign with a sort order may be used for Lookup Lists that support customised ordering. The *sortorder* value is added to the `SortOrder` entry for the Lookup List. A leading minus sign disables loading of the Lookup List, i.e. all values for that list will be skipped.

Where a hierarchy entry is specified, it is not necessary to add the upper levels of the hierarchy as these are added automatically. For example, if a text file contained:

```
Location|Australia|Victoria|Bendigo
```

then three entries are generated. The entries are the same as specifying:

```
Location|Australia
Location|Australia|Victoria
Location|Australia|Victoria|Bendigo
```

The use of text files provides a convenient mechanism for pre-loading Lookup Lists. As stated above, each entry loaded will have `Persistent` enabled, meaning the entry will persist even if it is not used. The only way to delete such an entry is to locate the record in the Lookup List module and delete it manually.

A System Administrator may configure a table to be skipped when Lookup Lists are being rebuilt. If a line of the form:

```
RELUTS=no
```

is found in a file called `vwoptions` in the database directory, `vwlutsrebuild` will not check data in that table.

The arguments to `vwlutsrebuild` is a list of Lookup List names to rebuild. If a list is not supplied, then all Lookup Lists are checked. The following examples cover some of the uses of `vwlutsrebuild`:

Rebuild all Lookup Lists

The command used to rebuild all Lookup Lists is:

```
vwlutsrebuild -t
```

It should not be necessary to run this command. The Lookup List Server maintains Lookup List synchronisation. The command may be used to ensure the Lookup List table is synchronised correctly.

Check Lookup List table is synchronised

The command used to check whether the Lookup List table is synchronised is:

```
vwlutsrebuild -d -t -v
```

This command runs the data generation phase only. The Lookup List table is not modified. The verbose option prints out any inconsistencies found.

Rebuild the Location and Admin Names Lookup List

The command used to rebuild the `Location` and `Admin Names` Lookup Lists is:

```
vwlutsrebuild Location 'Admin Names'
```

This command rebuilds the `Location` and `Admin Names` Lookup Lists. As the `-t` option is not specified, any entries in text files will not be loaded.

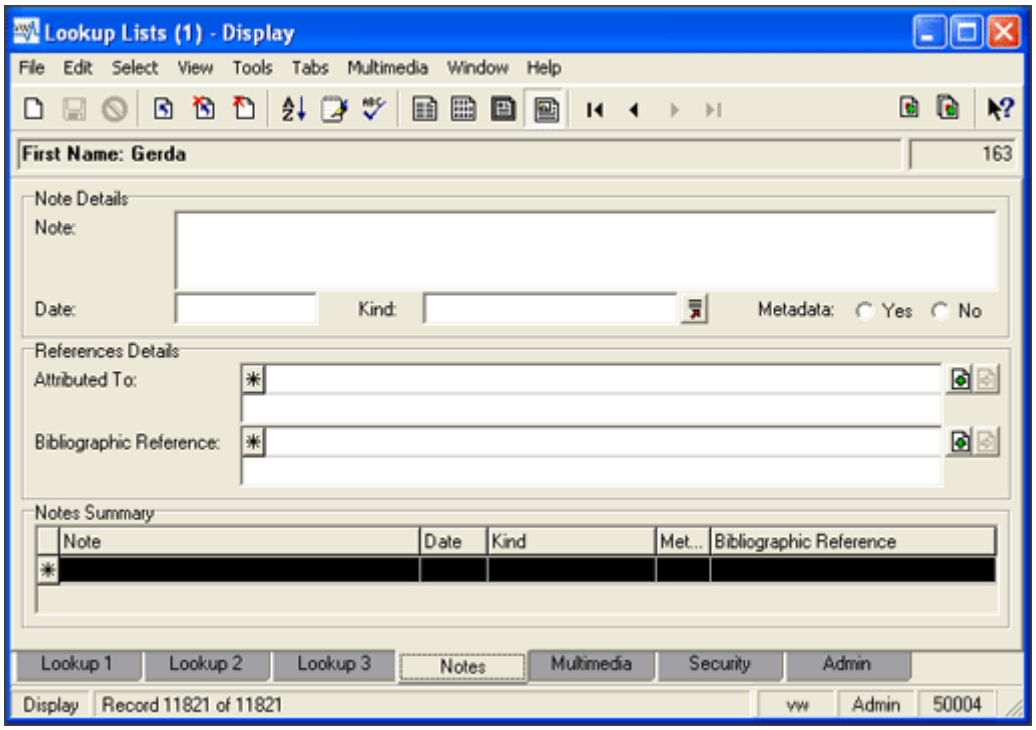
Note the use of single quotes to enclose the names of Lookup Lists that contain spaces.

Lookup List module

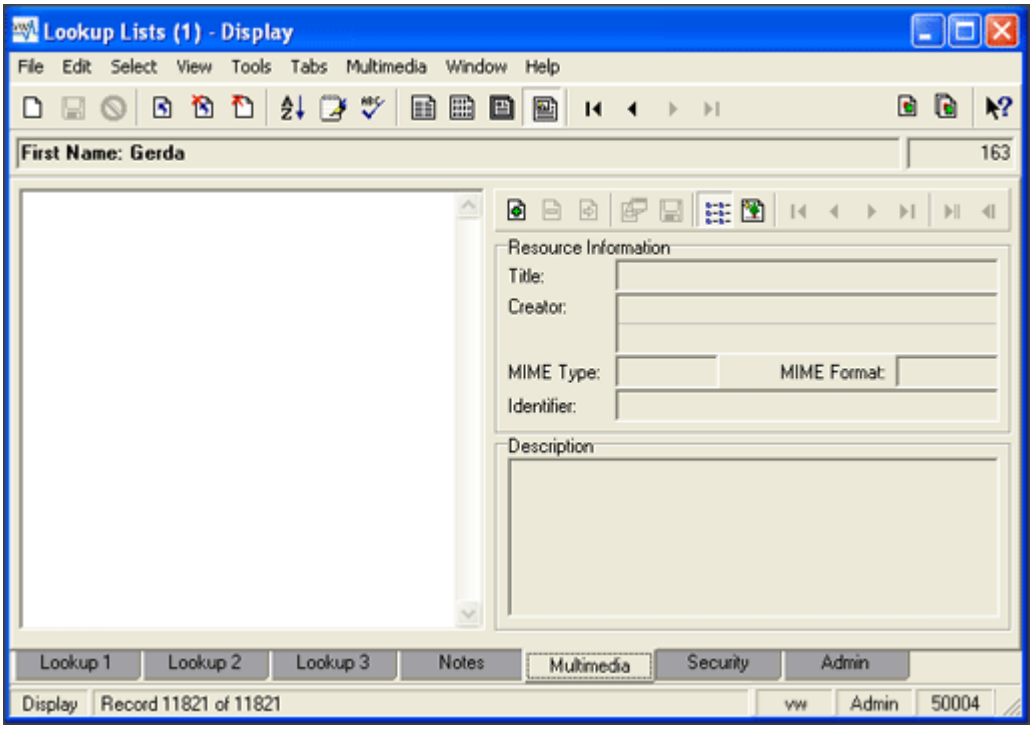
Until the release of the new Lookup List maintenance facilities in Vitalware 2.3.01, users could not interact with the Lookup List module and expect their changes to be preserved. The module itself was simplified to the point where only Lookup List specific information was stored. Fields available in all other modules were disabled. The maintenance changes introduced with Vitalware 2.3.01 mean that users can now access and use the module as they would any other module.

The following tabs are now available and functional in the Lookup Lists module:

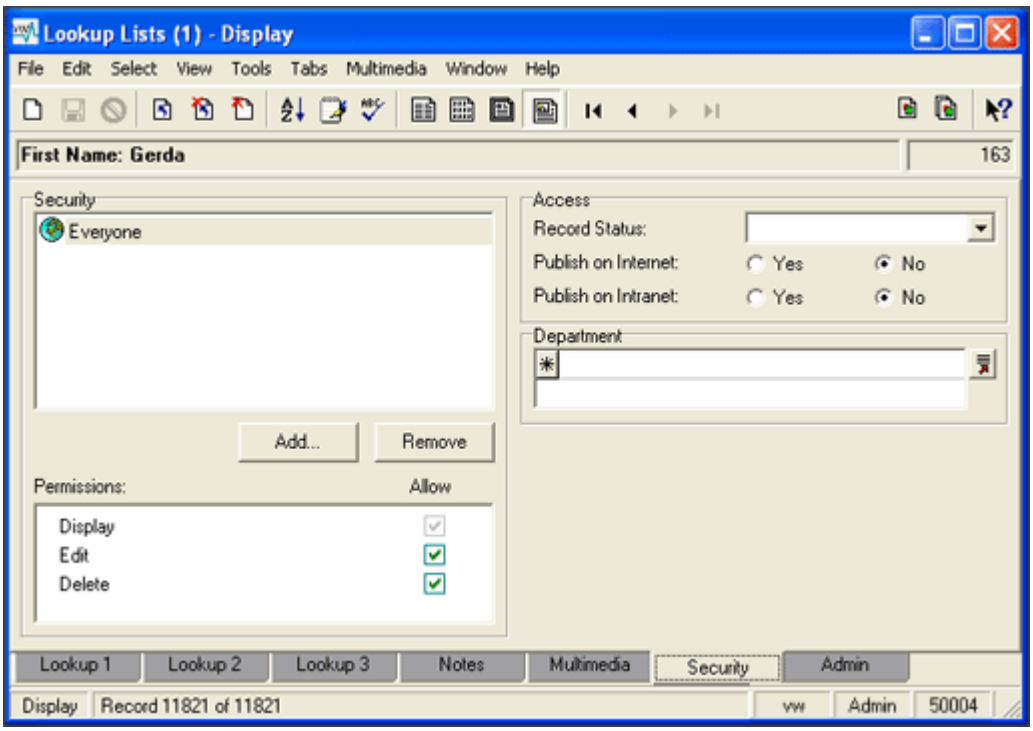
Notes



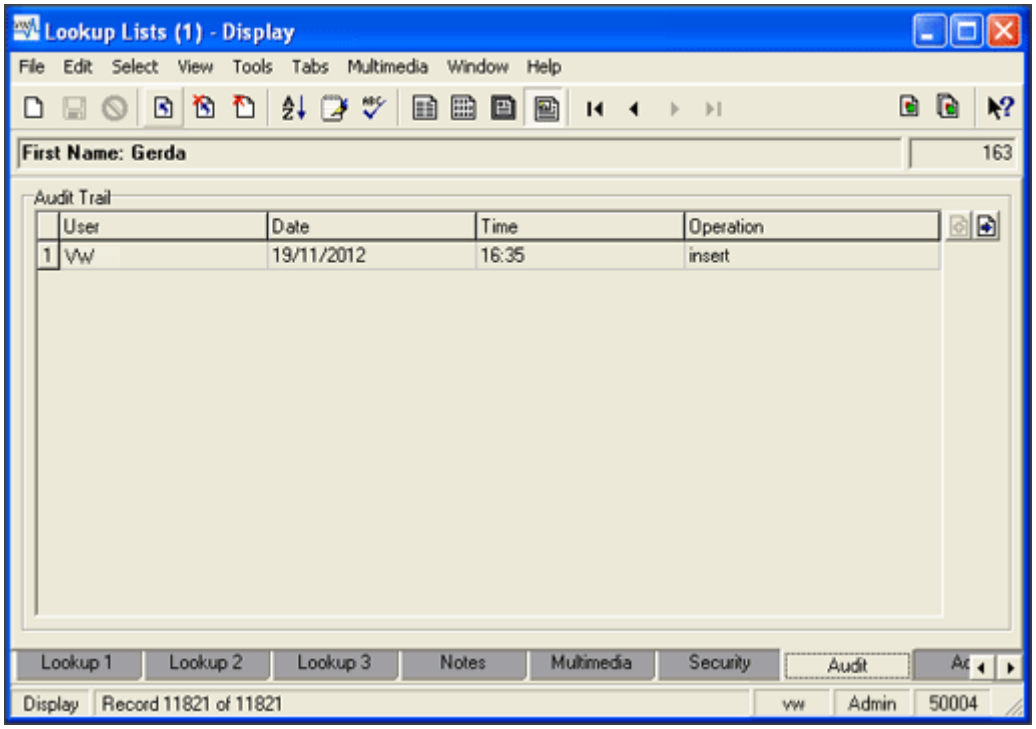
Multimedia



Security



Audit


Lookup Lists (1) - Display
File Edit Select View Tools Tabs Multimedia Window Help
First Name: Gerda
163
Audit Trail
User
Date
Time
Operation
1
VW
19/11/2012
16:35
insert
Lookup 1
Lookup 2
Lookup 3
Notes
Multimedia
Security
Audit
Display
Record 11821 of 11821
vw
Admin
50004

SECTION 2

Conclusion

The changes to the Lookup List maintenance framework made in Vitalware 2.3.01 provide a number of benefits:

- Lookup Lists are always up to date. Once the last use of a Lookup List value is deleted, the entry is removed from the Lookup List table.
- The Lookup List nightly maintenance procedure is no longer required. The removal of the maintenance decreases the amount of time required by the nightly maintenance routines.
- The Lookup List module now provides the standard tabs available in all other Vitalware modules.
- The `vwlutsrebuild` command may be used to check the consistency of the Lookup List table and apply any needed adjustments.

The changes to the Lookup List maintenance mechanism are the first of a series of changes designed to decrease the amount of time Vitalware requires to complete its maintenance runs.

Index

Vitalware®
Vital Records Management

The World's Premier Vital Records Software

Vitalware Documentation

The After Export facility

Document Version 1.1

Vitalware Version 2.0

KE Software

www.kesoftware.com

Contents

SECTION 1

Overview

The Scheduled Exports facility introduced with Vitalware 2.1.02 provides a mechanism for exporting data out of Vitalware on a regular or adhoc basis. Exports scheduled on a regular basis are executed by the server without any user intervention. When an export is complete, a record is added to the Exports module with:

- The results of the export
- A list of the files generated
- Any associated errors

The user must then retrieve the record from the Exports module to access the results and the exported data.

Introduced with Vitalware 2.2.02, the After Export facility allows a command to be executed once an export has completed. Amongst other things, the command can:

- Email the export files to a list of users.
- Email the results of the export to a list of users.
- Copy the export files to another machine behind a secure firewall.
- Copy the export files over the internet via a secure transfer mechanism.
- Send an SMS to a list of telephone numbers.

In fact, an After Export command may perform any number of tasks as it has full access to the Exports record generated. The command runs on the Vitalware server allowing access to the full facilities offered by the server. The After Export facility is designed to allow new commands to be added within the existing framework. In order to simplify the creation of new commands, the `KE::Export` perl module is provided; this incorporates much of the functionality required by an After Export command.

SECTION 2

Setting an After Export Command

A scheduled export is configured using the Export Properties dialogue in a module (by selecting **Tools>Export** from the module Menu bar). An After Export command is added to a scheduled export using the After Export tab of the Export Properties dialogue:

- If an After Export command has been defined, it can be selected from the *Command* drop list on the After Export tab.
- A command may require values to be provided by a user in order to run; if so, input boxes for the required values will display on the After Export tab when the command is selected from the *Command* drop list.

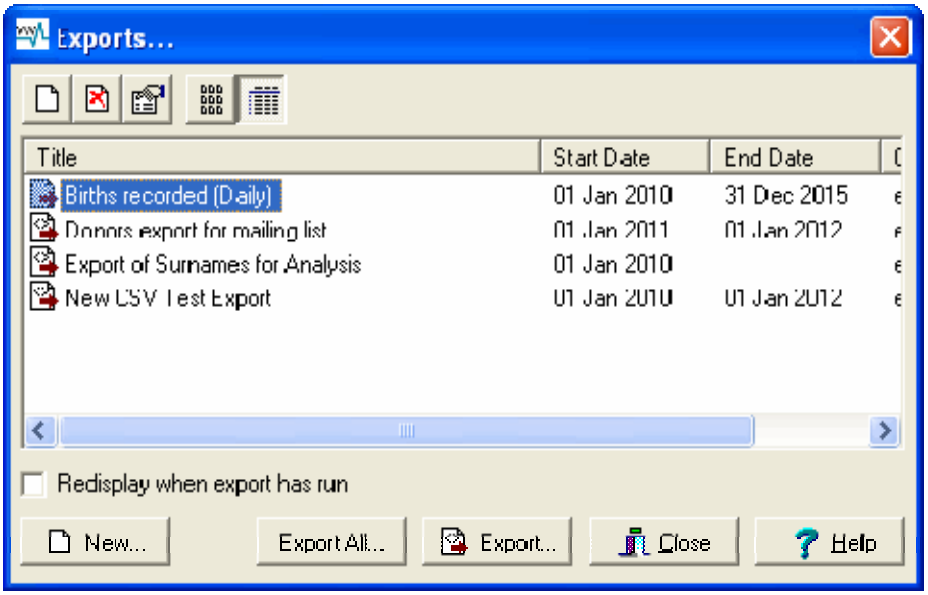
In Vitalware:

1. Open any module.
2. Search for or otherwise list a group of records.
3. Select **Tools>Export** in the Menu bar

   -OR-

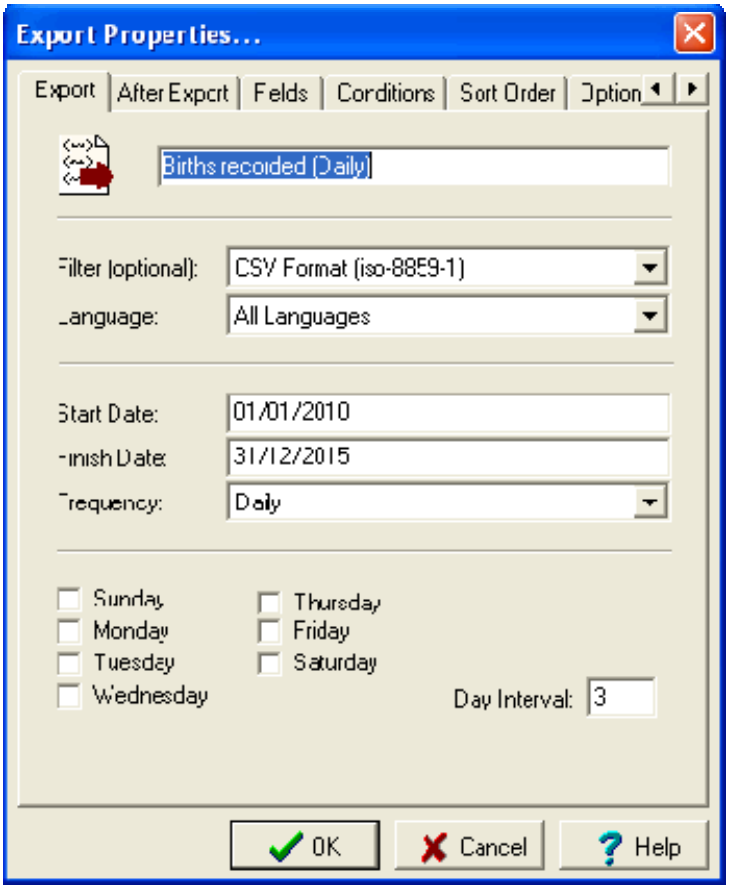
   Use the keyboard shortcut, ALT+T+X.

   The Exports dialogue displays with a list of scheduled exports for the current module:

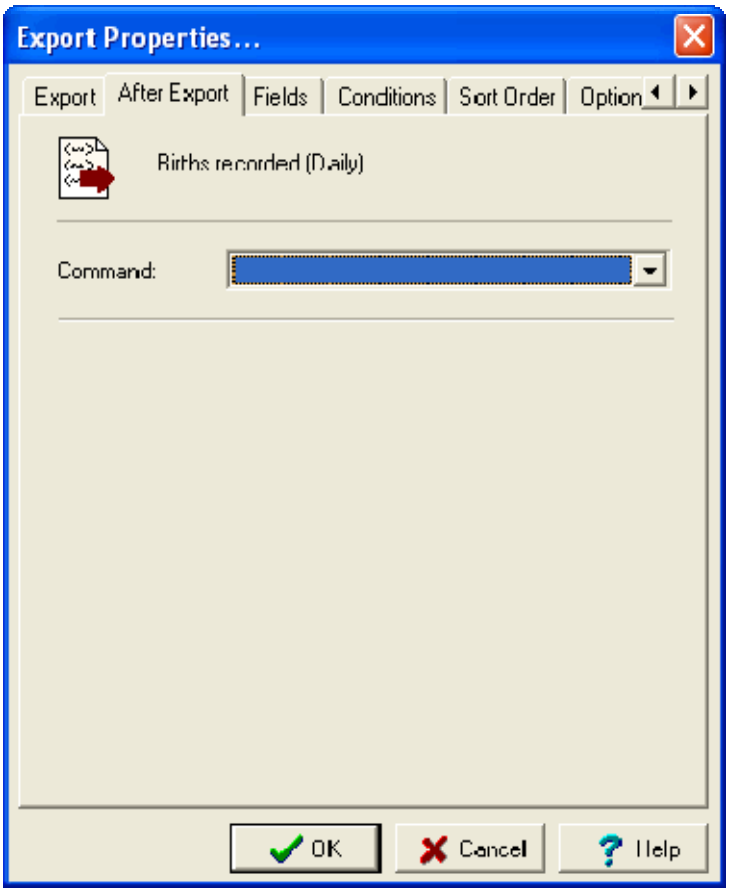


4. Select a scheduled export and click  .

The Export Properties dialogue displays:

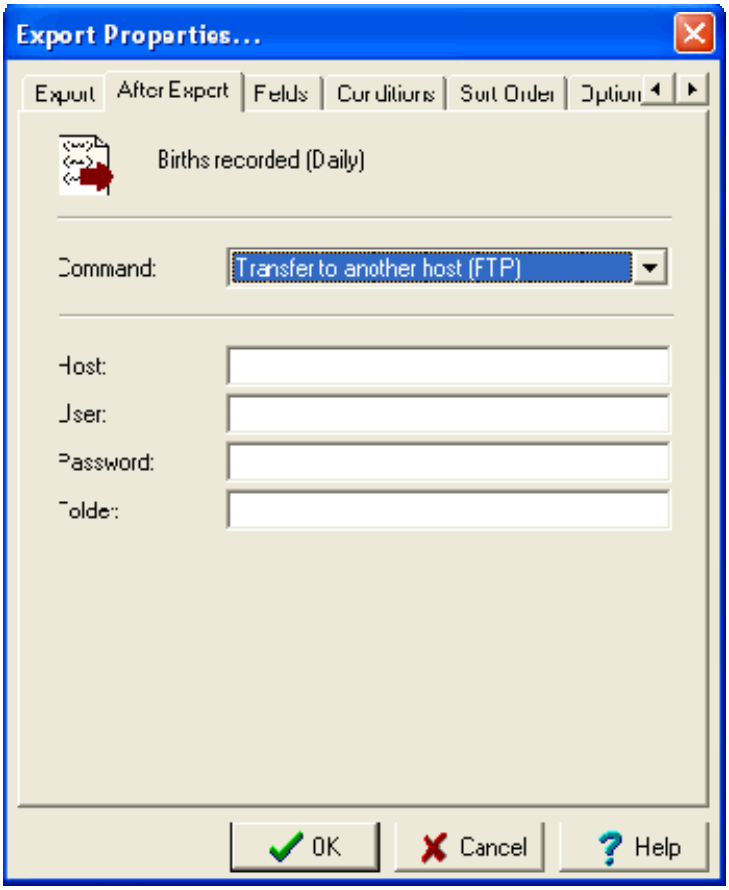


5. Select the **After Export** tab:



6. From the *Command* drop list, select the command to be executed.

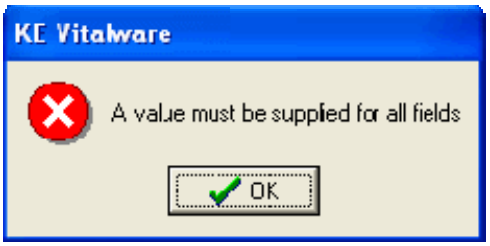
If the command requires values to be provided in order to run, input boxes for the parameters will display on the After Export tab:



7. Enter values for each of the fields and when you're done, click 

.

If a parameter was not provided, an error will display:



If all parameters have a value, the After Export command is saved and will be executed next time the scheduled export is run.

SECTION 3

Developing an After Export script

The After Export script

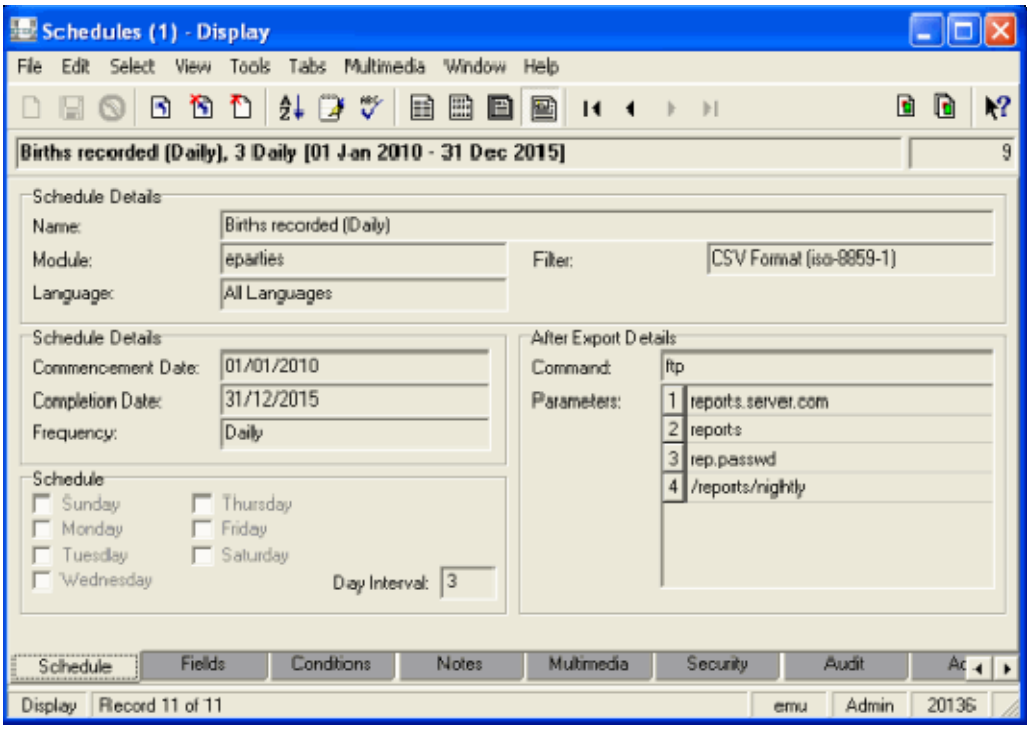
The Schedules module holds a record for each scheduled export defined in any module.

When an After Export command is added to a scheduled export, a script located on the Vitalware server is executed after the scheduled export is run. In the Schedules record for the scheduled export:

- *Command: (After Export Details)* holds the name of the script to be executed.
- *Parameters: (After Export Details)* lists the parameters required by the script.

In the Schedules record below, the script executed when the scheduled export is complete is called `ftp`. It has four parameters:

- `reports.server.com` (host name)
- `reports` (user name)
- `rep.passwd` (password)
- `/reports/nightly` (location)



The script file is located in one of the following directories on the Vitalware server:

- `local/etc/exports/after/`*table*
- `local/etc/exports/after`
- `etc/exports/after/`*table*
- `etc/exports/after`

where *table* is the name of the table (module) on which the export is performed. The table name is indicated in the *Module: (Schedule Details)* field (as shown above). To locate the script, the system looks for a file called `ftp` (in this case) in each of the directories listed, from first to last. When the file is found, the script stored in it is executed.

The hierarchy of directories listed above allows customised versions of existing scripts to be added by simply placing a file with the same name as the existing script into one of the `local` directories.

An After Export script is called by one of two methods:

1. In the first method, a script is called when a user selects a command from the *Command* drop list on the After Export tab of the Export Properties dialogue.

   In this case, the Vitalware client calls the script to get the title and list of parameters required by the command: the title displays in the *Command* drop list and the parameters are displayed below it. The Vitalware client calls the script with one argument, an option indicating which language to use to display the script's title and parameters. The usage is:

   ```
   script -lnum
   ```

   where *num* is a number corresponding to the language to use:

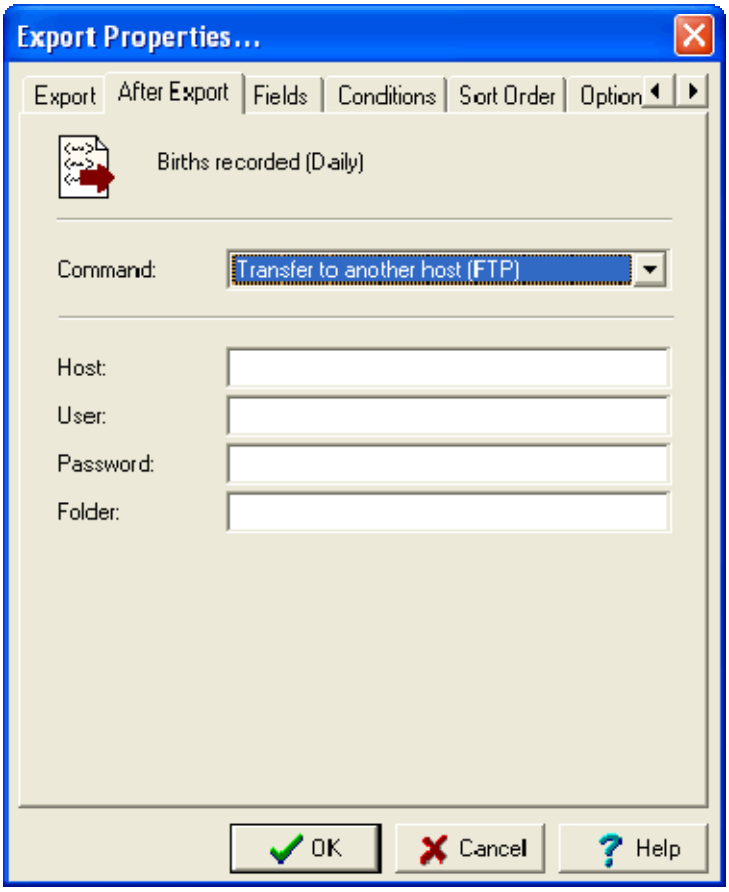
Number	Language
0	English
1	French
2	English (American)
3	Spanish
4	German
5	Italian
6	Dutch
7	Danish
8	Polish
9	Norwegian
10	Swedish
11	Greek
12	Arabic
13	Hebrew
14	French (Canadian)
15	Finish

The `l` in `script -l`*num* is a lowercase L.

For example, running `ftp - l0` produces:

```
Transfer to another host (FTP)
Host:
User:
Password:
Folder:
```

The first line is displayed in the *Command* drop list and the remaining lines are displayed as prompts for input boxes:



If the command is not supported by the local machine, then no output should be generated (the command should be hidden in the *Command* drop list) and a non-zero exit status returned.

2. In the second method, the back end `vwexport` command calls the After Export script after a scheduled export has run and a record with the results of the export has been created in the Exports module. In this instance the script is called with the IRN (Internal Record Number) of the record created in the Exports (eexports) module. The usage is:

   `script` *exportirn*

   The script should use *exportirn* to access the Exports (eexports) record and perform whatever activities the script was designed to do (e.g. send an email, copy files, etc.). If an error occurs while processing the export, an error message should be printed and a non-zero exit status returned. If the script completes successfully, a zero exit status should be returned.

These two methods are explained in detail in *Creating an After Export script* (page 12).

The perl code below is a sample `ftp` script:

```
#!/usr/bin/perl

#
#  Copyright (c) 1998-2011 KE Software Pty Ltd
#

use strict;
use KE::Export;

#
#  Parameters for ftp.
#
my $prompts =
{
        0       => [
                        "Transfer to another host (FTP)",
                        "Host:",
                        "User:",
                        "Password:",
                        "Folder:"
                   ]
};

#
#  Check whether ftp is supported on this machine.
#
my $ftp = KE::Export::Ftp->new();
if (! $ftp->IsSupported())
{
        exit 1;
}

#
#  Parse the arguments.
#
my $export = KE::Export->new();
if (! $export->ParseArgs(\@ARGV))
{
        exit 1;
}

#
#  List parameters if required.
#
if ($export->ListParameters($prompts))
{
        exit 0;
}

#
#  Do the transfer with the required arguments.
#
my $status = $ftp->Execute
(
        host            =>      $export->GetData('Parameters_tab')->[0],
        user            =>      $export->GetData('Parameters_tab')->[1],
        password        =>      $export->GetData('Parameters_tab')->[2],
```

```
        destination     =>      $export->GetData('Parameters_tab')->[3],
        filelist        =>      $export->GetData('FileName_tab')
);

#
#  Send back the error status.
#
exit $status;
```

This script uses the perl `KE::Export` module which provides most of the required functionality.

In essence the script:

1. Checks whether the server provides FTP support. If not, it returns a non-zero exit status (i.e. `1`).
2. Parses the arguments to determine which version of the script has been called.

   If the arguments are invalid, a non-zero exit status is returned once again.

   -OR-

   If a valid `-l`*num* argument was given and the script was called using the first method described above, the script prints out the title and parameters for language *num*. Then exits with a zero exit status (indicating success).

   -OR-

   If the script was called using the second method described above, the file transfer is performed using ftp. An ftp object, passed the required parameters, transfers the files. The status of the ftp object is used for the exit status, where a non-zero value indicates the transfer failed, and a zero indicates success.

For a complete description of the functionality provided by the `KE::Export` module see *KE::Export usage*.

Creating an After Export script

The script for an After Export command must handle the two usage cases where the script is either called with:

- `-l`*num* to provide the command title and list of parameters

  -OR-

- with the IRN of an eexports record to perform what the script is required to do

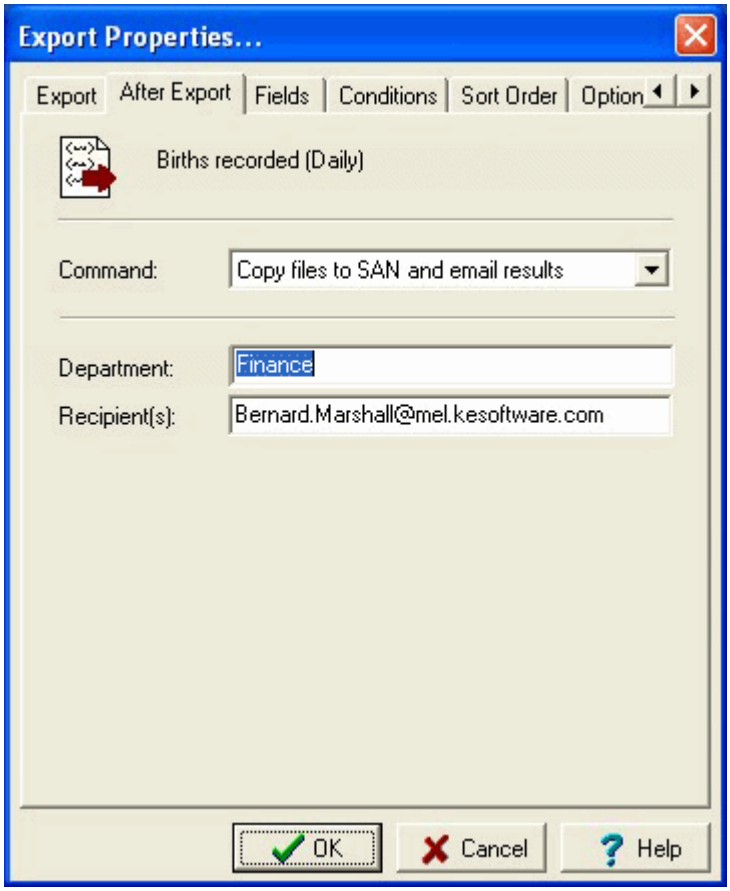
In order to make script writing easier, a perl module `KE::Export` is provided that encompasses most of the functionality needed by an After Export script. It is recommended that you use the module to cut down development time. For a complete description of the functionality provided by the `KE::Export` module see *KE::Export usage*.

To demonstrate how a new command could be put together we'll write a script that copies the output files from an export into a location on a SAN drive (mounted as `/exports`) based on the date of the export and a user specified department and then send an email to a user supplied address.

The first part of the script involves setting up the title and parameters. Two parameters are required, the first is the department under which to file the export files and the second is the email addresses to notify once the files have been copied:

```
#
#  Parameters for copy and email notification
#
my $prompts =
{
        0       => [
                        "Copy files to SAN and email results",
                        "Department:",
                        "Recipient(s):"
                   ]
};
```

The first string is the title to show in the *Command* drop list, and the following two parameters allow the department and email addresses to be entered. The resulting After Export tab is:



Next check to make sure the server supports both the copying of files and emailing of messages. If support for either is not provided, the command should be hidden in the *Command* drop list. An exit status of `1` indicates the command is not supported:

```
#
#  Check whether email and copy are supported on this machine.
#
my $copy = KE::Export::Copy->new();
my $email = KE::Export::Email->new();
if (! $email->IsSupported() || ! $copy->IsSupported())
{
        exit 1;
}
```

Once we have confirmed the required functionality is supported, we check that the supplied arguments are valid. If they are not, exit with an error status of `1`:

```
#
#  Parse the arguments.
#
my $export = KE::Export->new();
if (! $export->ParseArgs(\@ARGV))
{
        exit 1;
}
```

Since the supplied arguments are acceptable, look for the `-l`*num* case. If the arguments match, the prompts defined above are printed out and we exit with a successful status of `0`:

```
#
#  List parameters if required.
#
if ($export->ListParameters($prompts))
{
        exit 0;
}
```

If we get to here, we must process the `eexports` record whose IRN was supplied as the argument. First build up the full path to the folder in which we want to store the export files. We use the *FileRunDate* column on the eexports record to get the date on which the export was run, and the first entry in the *Parameters_tab* column list to get the department entered by the user. Once we have the destination path, make sure the folder exists (using `mkdir`). If the folder cannot be created, exit with an error status of `1`:

```
#
#  Build up the destination directory and make sure it exists.
#
my $date = $export->GetData('FileRunDate');
my $department = $export->GetData('Parameters_tab')->[0];
my $destination = "/tmp/$department/$date";
if (system("mkdir -p '$destination'") != 0)
{
        exit 1;
}
```

It is now time to copy the export files to the destination folder. We use a `KE::Export::Copy` object to perform the transfer. The *FileName_tab* column contains a list of all the files created by the export:

```
#
#  Do the copy with the required arguments.
#
my $status = $copy->Execute
(
        destination     =>      $destination,
        filelist        =>      $export->GetData('FileName_tab')
);
```

Now build up the body of the email message to send to the recipients. Include the name of the export, the date on which it ran, the folder into which the export files were copied and indicate whether the transfer was successful:

```
#
#  Build up the email message to send
#
my $body = "The files from export \"" .
           $export->GetData('ScheduleRef:eschedule:Name') .
          "\", run on $date have been copied to $destination.\n" .
          "The copy " . $status ? "failed" : "succeeded.\n";
```

Finally, send the email message to all the recipients. Use the second parameter to extract the email addresses of the recipients. The email's subject is the name of the scheduled export:

```
#
#  Do the email with the required arguments.
#
my $status = $email->Execute
(
        recipients      =>      $export->GetData('Parameters_tab')->[1],
        subject         =>      $export-
>GetData('ScheduleRef:eschedule:Name'),
        body            =>      $body
);
```

Return the status of the email object, indicating whether the emails were sent correctly or not:

```
#
#  Send back the error status.
#
exit $status;
```

The complete script is:

```
#!/usr/bin/perl

#
#  Copyright (c) 1998-2011 KE Software Pty Ltd
#

use strict;
use KE::Export;

#
#  Parameters for copy and email notification
#
my $prompts =
{
        0       => [
                        "Copy files to SAN and email results",
                        "Department:",
                        "Recipient(s):"
                   ]
};

#
#  Check whether email and copy are supported on this machine.
#
my $copy = KE::Export::Copy->new();
my $email = KE::Export::Email->new();
if (! $email->IsSupported() || ! $copy->IsSupported())
{
        exit 1;
}

#
#  Parse the arguments.
#
my $export = KE::Export->new();
if (! $export->ParseArgs(\@ARGV))
{
        exit 1;
}
```

```
#
#  List parameters if required.
#
if ($export->ListParameters($prompts))
{
        exit 0;
}

#
#  Build up the destination directory and make sure it exists.
#
my $date = $export->GetData('FileRunDate');
my $department = $export->GetData('Parameters_tab')->[0];
my $destination = "/tmp/$department/$date";
if (system("mkdir -p '$destination'") != 0)
{
        exit 1;
}

#
#  Do the copy with the required arguments.
#
my $status = $copy->Execute
(
        destination     =>      $destination,
        filelist        =>      $export->GetData('FileName_tab')
);

#
#  Build up the email message to send.
#
my $body = "The files from export \"" .
            $export->GetData('ScheduleRef:eschedule:Name') .
           "\", run on $date have been copied to $destination.\n" .
           "The copy " . ($status ? "failed" : "succeeded") . ".\n";

#
#  Do the email with the required arguments.
#
my $status = $email->Execute
(
    recipients    =>    $export->GetData('Parameters_tab')->[1],
    subject       =>    $export->GetData('ScheduleRef:eschedule:Name'),
    body          =>    $body
);

#
#  Send back the error status.
#
exit $status;
```

The script should be placed in the directory `local/etc/exports/after` as it is a customised script. The file name of the script is not important, but something like `copyemail` would be appropriate. Remember to change the file permissions so it can be executed (i.e. `chmod 755 copyemail`). Your script is now ready for use.

KE::Export usage

The `KE::Export` perl package provides a number of very useful objects that simplify the process of creating an After Export script. The package file is located in `utils/KE/Export.pm` and is documented fully. To view the documentation use `pod2text Export.pm` (assuming you are in the `utils/KE` directory). For your convenience the documentation is reproduced here:

```
NAME
       KE::Export - A set of objects usable by After Export scripts

SYNOPSIS
```

```
use KE::Export;
my $prompts =
  {
     0    => [
                  'Copy to another folder (CP)',
                  'Folder:'
             ]
  };

  my $copy = KE::Export::Copy->new();
  if (! $copy->IsSupported())
  {
        exit 1;
  }

  my $export = KE::Export->new();
  if (! $export->ParseArgs(\@ARGV))
  {
        exit 1;
  }
  if ($export->ListParameters($prompts))
  {
        exit 0;
  }

  my $status = $copy->Execute
  (
     destination  =>   $export->GetData('Parameters_tab')->[0],
     filelist     =>   $export->GetData('FileName_tab')
  );

  exit $status;
```

```
DESCRIPTION
       The "KE::Export" module provides a set of objects to make the
       implementation of After Export commands easier. An After Export
       command may be registered with a scheduled export through the "After
       Export" tab in the Export Properties dialogue box. If an After
       Export command is registered with a scheduled export, the command is
       executed once the export phase is complete.
```

The After Export command provides a mechanism for dealing with the exported data after it is generated. In particular the command may:

* Email the results of the export to a list of users.

* Email the export files to a list of users.

* Copy the export files onto another machine.

* Send an SMS to a list of telephone numbers.

An After Export command corresponds to a script located in one of the following directories on the server machine:

local/etc/exports/I<table>/after
local/etc/exports/after
etc/exports/I<table>/after
etc/exports/after

The directories are examined in the order specified above to locate the required script. Using this mechanism it is possible to override a script provided with the system with a custom built one. To do so just add your custom script, with the same name as the script you are overriding, to a directory listed above the directory in which the system script is located. For example, if you want to override the system "ftp" script stored in etc/export/after/ftp, you would place your script in the file local/etc/export/after/ftp.

Each After Export script may be invoked in two ways. These are:

"script -l*num*"

where *num* is the language number to use when outputting the parameters. This form of the script is expected to print out the *title* of the script to use in the drop list on the "After Export" tab in the client and a list of required parameter prompts, one per line. For example, the "ftp" script invoked by "ftp -l0" (to print out the title and parameters in English) produces:

Transfer to another host (FTP)

Host:

User:

Password:

where the first line is displayed in the "Command:" drop list on the "After Export" properties tab and the remaining lines are shown as input boxes below the "Command:" drop list with the text used as the prompt. The user must specify each of the parameters before a valid After Export command may be saved.

The language number supplied via the "-l" option determines the language in which the output should appear. The registered language numbers are:

0 - English

1 - French

2 - English (American)

3 - Spanish

4 - German

5 - Italian

6 - Dutch

7 - Danish

8 - Polish

9 - Norwegian

10 - Swedish

11 - Greek

12 - Arabic

13 - Hebrew

14 - French (Canadian)

15 - Finnish

"script *exportirn*"

The second way of invoking a script is to supply the irn (Internal Record Number) of the record in the "eexports" table on which the script is to operate. In this case the script needs to perform what is deemed its duty. For example, in the case of the "ftp" script, the export files produced will be FTPed to another host, The username, password and destination on the remote host are used to make the transfer.

The normal life cycle of a "KE::Export" object is:

1 Create the object (via "new()").

2 Parse any script parameters to determine which of the two uses of the script is appropriate (via "ParseArgs()").

3 Output the title and parameters if the script was invoked with the first usage (via "ListParameters()").

4 Extract the parameters and execute the required functionality if the script was invoked with the second usage (via "GetData()").

For examples of how to use the "KE::Export" module please review the After Export scripts installed on the server machine with the default installation.

KE::Export

A "KE::Export" object provides a wrapper around a set of utility functions. These functions are designed to simplify the process of writing After Export scripts by encapsulating standard functionality. In particular, the following facilities are offered:

* A standard way of parsing the script's arguments to determine which type of invocation was used. See ParseArgs().

* A standard mechanism for outputting the script parameters when invoked with the "-l" option. See ListParameters().

* A mechanism for determining the languages supported by the server. See Languages().

* A means of extracting data from the underlying batch record and its associated schedule record. In fact, any links from the batch record may be followed to retrieve data from other

modules. See GetData().

Each After Export script should use a "KE::Export" object to simplify access to the underlying export record.

Methods

new()

```
$export = KE::Batch->new();
```

Creates an object that provides access to the underlying export record. The object also provides access to helper functions that simplify After Export scripts. In order to release all resources associated with a "KE::Export" object (for example, a connection to a server) "undef" should be assigned to the object variable once the object is no longer required.

GetData($colname)

```
$data = $export->GetData('StartDate');
$host = $export->GetData('Parameters_tab')->[0];
$filelist = $export->GetData('FileList_tab');
$name = $export->GetData('ScheduleRef:eschedule:Name');
```

Retrieves the data for the given column. The $colname argument may be the name of any column in the "eexports" table. The value returned is consistent with the kind of data stored in the column. An atomic column returns a string, a table returns a reference to a list of strings and a nested table returns a reference to a list where each element is itself a list of strings.

It is also possible to access columns in other modules by specifying the name of the link field in "eexports" followed by the table name and column in the linked table. Each component is separated by a colon. There is no limit to the number of components in the column name for linked fields. For example, the column name "ScheduleRef:eschedule:Name" uses the link from "eexports" to "eschedule" (via column ScheduleRef) to access the Name field in the "eschedule" table. In other words, the name of the scheduled export is retrieved.

The "Parameters_tab" column provides access to the command parameters entered for the After Export command. The "FileList_tab" column provides a list of all the files generated by the export process.

Languages()

```
$langlist = $export->Languages();
```

Retrieves the list of languages supported by the server. The list consists of a string of semi-colon separated language numbers. For example, the string "0;1" indicates the server supports English and French, where English is the primary language. The "Languages()" call is used by After Export commands where text is generated as part of the script. The text must be output in languages supported by the server.

The "Languages()" function returns the value of the "System|Setting|Language|Supported" Registry entry.

ListParameters($prompts)

```
my $prompts =
      {
         0    => [
                        "Email export results (SMTP)",
                        "Recipient(s):"
                     ]
      };

      if ($export->ListParameters($prompts))
      {
         exit 0;
      }
```

Prints out the title and parameters for the After Export command. If the "ParseArgs()" call determined the list of parameters should be printed (via the "-lnum" option), then the title and parameters for the given language number are printed and the call returns 1. If the "-lnum" option was not specified, the call returns 0.

The $prompts argument is a reference to a hash, where the key is the language number and the value is a reference to a list of strings. The first string is the title and subsequent strings are parameters. The title string is shown in the Command drop list, and the parameters strings are shown below the Command drop list with a corresponding data entry field where values may be specified.

ParseArgs(\@ARGV)

```
if (! ParseArgs(\@ARGV))
      {
         exit 1;
      }
```

Parses the on-line arguments determining whether the parameters are to be printed (as "-lnum" was supplied) or the command executed (the eexports irn was supplied).

If invalid options are found, a usage message is printed and 0 is returned. If the arguments are correct, 1 is returned.

KE::Export::Command

The "KE::Export::Command" class is the base class for all After Export commands. It consists of two methods each After Export command must implement. The first is "IsSupported()" which returns 1 if the After Export command is supported by the server. Some commands may require special software or certain perl modules to be installed before they can be used. The second method is "Execute()", which performs the command itself.

This class should not be called directly from within After Export scripts, rather a sub-class should be created and the IsSupported() and Execute() methods overridden.

Methods

new()

```
$export = KE::Batch::Command->new();
$export = KE::Batch::Command->new(debug => 1);
```

Creates an object used to execute an After Export command. The "new()" method should not be called directly, rather sub-

classed versions should be used. Debugging may be enabled by setting the named argument "debug" to a non-zero value.

Execute()

```
$status = $command->Execute()
```

Executes the After Export command. The "Execute()" method implements the functionality required by the After Export command. For example, if the command is to email the resulting export files to a user, the method must perform the actual emailing and attaching of export files.

The method should return 0 if the command was successful, otherwise 1 should be returned.

Each sub-class must override this method to implement the functionality specific to the sub-class's command.

IsSupported()

```
$support = $command->IsSupported();
```

Determines whether the After Export command is supported by the server. An After Export command may have dependencies on a number of programs or perl modules. The "IsSupported()" method checks each dependency is available, and if so returns 1, otherwise 0. A return value of 0, removes the After Export command from the Command drop list in the client.

KE::Export::Sftp

The "KE::Export::Sftp" class allows secure file transfer (SFTP) to be used to copy the export files to another machine. The "sftp" functionality provided by the "scp" command set is used for the file transfers. The file is encrypted during the copy ensuring privacy of data.

Methods

```
Execute()
      $sftp = KE::Export::Sftp->new();
      $status = $sftp->Execute
      (
         host         =>   'other.machine',
         user         =>   'username',
         password     =>   'passwd',
         destination  =>   '/exports/nightly/',
         filelist     =>   $export->GetData('FileList_tab')
      );
```

Performs a secure copy of the export files generated to another host. A number of named arguments are available, all are mandatory:

host

The host name of the machine onto which the export files are to be copied.

user

The user name to use to log in to the remote machine.

password

The password to use to log in to the remote machine.

destination

The directory in which the export files are to be placed. The directory must exist.

filelist

A reference to a list containing the files to be transferred to the remote machine.

"Execute()" returns 0 if the transfers succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$sftp = KE::Export::Sftp->new();
$status = $sftp->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide secure file transfer. A return value of 0 implies the server does not support secure file transfer, while a value of 1 implies it does.

KE::Export::Ftp

The "KE::Export::Ftp" class allows file transfer (FTP) to be used to copy the export files to another machine. The data transferred is not encrypted while in transit. As the password is sent to the server without any encryption, that is as clear text, "KE::Export::Ftp" should be used only within internal networks, behind a secure firewall. FTP may offer superior throughput to SFTP.

Methods

```
Execute()
      $ftp = KE::Export::Ftp->new();
      $status = $ftp->Execute
      (
         host         =>   'other.machine',
         user         =>   'username',
         password     =>   'passwd',
         destination  =>   '/exports/nightly/',
         filelist     =>   $export->GetData('FileList_tab')
      );
```

Performs a copy of the export files generated to another host. A number of named arguments are available, all are mandatory:

host

The host name of the machine onto which the export files are to be copied.

user

The user name to use to log in to the remote machine.

password

The password to use to log in to the remote machine.

destination

The directory in which the export files are to be placed. The directory must exist.

filelist

A reference to a list containing the files to be transferred to the remote machine.

"Execute()" returns 0 if the transfers succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$ftp = KE::Export::Ftp->new();
$status = $ftp->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide file transfer support. A return value of 0 implies the server does not support file transfer, while a value of 1 implies it does.

KE::Export::Scp

The "KE::Export::Scp" class allows secure copy (SCP) to be used to transfer the export files to another machine. The data transferred is encrypted while in transit. An SCP connection may be formed by either supplying a password, or not. If a password is not supplied, X509 based certificates may be used removing the need for a password. In general, X509 based connections are preferred as a password does not need to be stored in the command script.

Methods

```
Execute()
      $scp = KE::Export::Scp->new();
      $status = $scp->Execute
      (
         host         =>   'other.machine',
         user         =>   'username',
         password     =>   'passwd',
         destination  =>   '/exports/nightly/',
         filelist     =>   $export->GetData('FileList_tab')
      );
```

Performs a copy of the export files generated to another host. A number of named arguments are available, most are mandatory:

host

The host name of the machine onto which the export files are to be copied. A host name must be supplied.

user

The user name to use to log in to the remote machine. A user name must be supplied.

password

The password to use to log in to the remote machine. If a password is not supplied, an X509 certificate based connection is attempted.

destination

The directory in which the export files are to be placed. The directory must exist. A destination must be supplied.

filelist

A reference to a list containing the files to be transferred to the remote machine. A list of files to transfer must be supplied.

"Execute()" returns 0 if the transfers succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$scp = KE::Export::Scp->new();
$status = $scp->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide secure file copying. A return value of 0 implies the server does not support file copying, while a value of 1 implies it does.

KE::Export::Copy

The "KE::Export::Copy" class allows the export files to be copied to another location on the same machine. The command may also be used to copy the export files onto a file system mounted on the same machine (e.g. a SAMBA share).

Methods

```
Execute()
      $copy = KE::Export::Copy->new();
      $status = $copy->Execute
      (
          destination  =>  '/exports/nightly/',
          filelist     =>  $export->GetData('FileList_tab')
      );
```

Performs a copy of the export files generated to another location on the same host. Two named arguments are available, both are mandatory:

destination

The directory in which the export files are to be placed. The directory must exist.

filelist

A reference to a list containing the files to be copied to another location.

"Execute()" returns 0 if the copy succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$copy = KE::Export::Copy->new();
$status = $copy->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide file copying. A return value of 0 implies the server does not support file copying, while a value of 1 implies it does.

KE::Export::Email

The "KE::Export::Email" class allows an email notification to be sent to a list of email addresses when a scheduled export is complete. If the list of export files is provided, the files are sent as an attachment to the notification email, otherwise just the result of the export is emailed.

Methods

```
Execute()
   $email = KE::Export::Email->new();
   $status = $email->Execute
   (
       recipients   =>   'user1@abc.com, user2@def.com',
       filelist     =>   $export->GetData('FileList_tab')
   );
```

Emails the results of a scheduled export to a list of users. If the export files are supplied, via the filelist named parameter, the export files are attached to the email. Two named arguments are available, one of which is mandatory:

recipients

A comma separated list of email addresses defining users who should receive the results of the scheduled export. At least one recipient must be supplied.

filelist

A reference to a list containing the files to be attached to the email message. If a filelist is not supplied, only the results are emailed to each user.

"Execute()" returns 0 if the email succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$email = KE::Export::Email->new();
$status = $email->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide email services. A return value of 0 implies the server does not support emailing, while a value of 1 implies it does.

Index