User Guide

for

THE VISUAL DUMP ANALYZER

GUI

Version 5.0

© 2008 Visual Data ApS. All Rights Reserved

To receive further information about **Visual Data** products please contact:

Visual Data ApS
Attn: Lars Kruse Bøggild
Hvidtjørnen 36
DK-2791 Dragør / CPH
Denmark

Tel: +45 32 94 31 94

e-mail: support@visualdata.info
Internet: www.visualdata.info

Visual Data ApS
Attn: Gary J. Sowada
12181 Highway 27
Little Falls, MN USA 56345

Tel: +1 320.632.6200
Fax: +1 320.632.6240

e-mail: Sales@visualdata.info
Internet: www.visualdata.info

© 2008 Visual Data ApS. All Rights Reserved

User Guide for The Visual Dump Analyzer

© 2008 Visual Data ApS. All Rights Reserved

Overview

The Visual Dump Analyzer is an innovative debug tool developed for the Unisys USAS environment. It is designed to simplify and expedite the complicated process of solving a mainframe dump (abort) by intuitively displaying virtually all relevant information in an easy-to-use graphical environment, all with point-and-click and drag-and-drop ease.

Focused on assisting both skillful and novice application programmers alike, The Visual Dump Analyzer eliminates the tedious and error-prone task of converting symbolic variables and lines of code to their absolute addresses within the dump. That is, instead of spending considerable time scouring over a raw octal dump to find a variables value, you can now simply click on the variable within the displayed source code or reference it within an alphabetized table to quickly determine its value. Lines of source code that result in abnormal execution (contingencies) are also quickly recognized as they are immediately displayed and explained making for fast and accurate solutions.

The Visual Dump Analyzer will also assist you in solving a dump by graphically displaying the complete instruction and data memory layouts, by presenting all related source code and data buffers in a variety of formats, by describing the nature of the dump in a natural language, and by displaying all related information stored within its knowledge database.

Once a dump is solved, you can add this vital information to The Visual Dump Analyzer Knowledge Database so the next time a similar abort occurs, the notes and hints will be immediately available for yourself or your colleagues. In effect, the value of this tool increase with time for critical details that are easily forgotten will always be available for automatic reference.

The Environment

The Visual Dump Analyzer resides on a PC having the MS Windows 95/98/NT/2000/XP/VISTA operating system.

When you enter the name of a dump that you wish to analyze, all data associated with this dump is automatically downloaded from the mainframe with virtually all further processing taking place on the PC.

Realizing some programmers may not use it on a very frequent basis, The Visual Dump Analyzer strives to simplify the complex task of dump analysis by using an intuitive design that does not require memorization of commands or tricks that are only gained through experience.

The Visual Dump Analyzer uses many of the standard MS Windows functions such as scrolling, resizing windows, copy-and-paste, drag-and-drop, printing, and context menus. Therefore, this familiar environment will allow you to effectively use the product from the very outset.

Symbolic debugging

The Visual Dump Analyzer will retrieve all information necessary to thoroughly analyze a dump: the related program source code, program absolutes (executables), file and packet record layouts, and the dump itself. The program absolute will be used as the glue between the source code and the dump, thus enabling you to reference variables and source line numbers rather than their absolute addresses (i.e., symbolic debugging). This powerful feature eliminates your need to know where local variables, instructions, and global data are located within the dump thus freeing yourself to concentrate on the actual task at hand – solving the dump.

Configuration

The first time you use The Visual Dump Analyzer, it will guide you through a simple dialog that will download all the necessary configuration information from one of the mainframes. Even though you plan to use The Visual Dump Analyzer towards more mainframes, you only have to configure it once as the downloaded configuration contains information for all mainframes. (See ***Getting Started – Auto Config***)

Communication setup

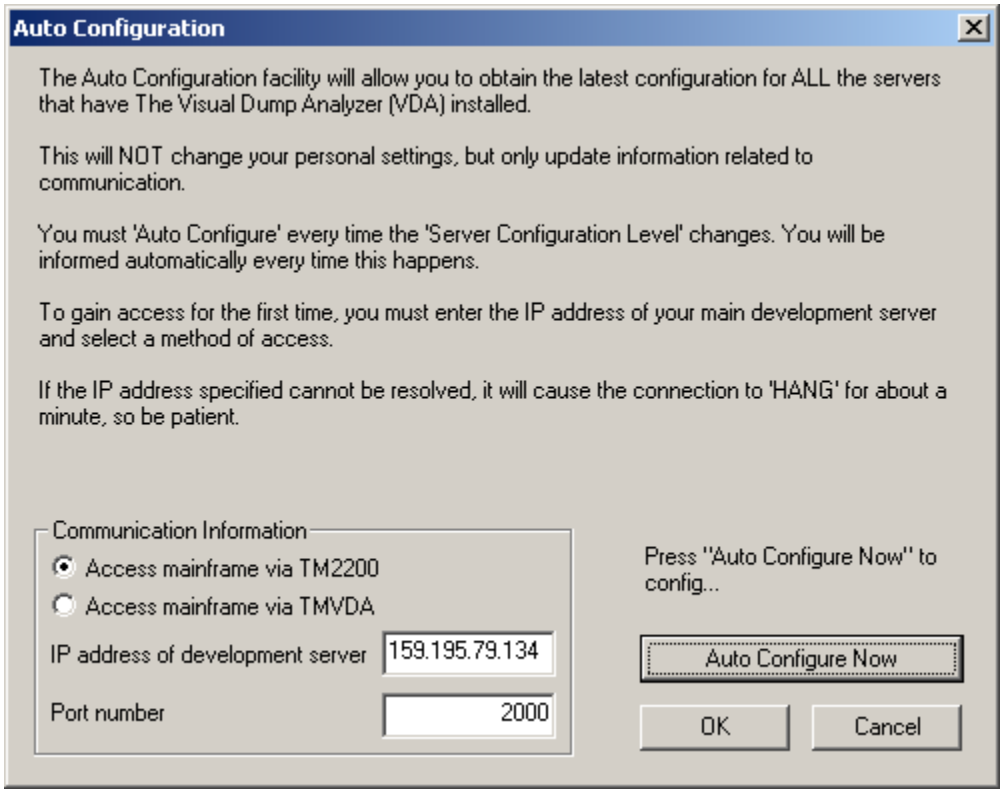
The communication tool that is used between the client (VDA.exe) and the server can be either The Visual Dump Analyzer's own Transaction Manger (TMVDA) or it can be the Unisys Open/Env (OLTP) Transaction Manager (TM2200). TMVDA uses normal TCP/IP communication between the client workstations and TMVDA on the mainframe.

Independent of the communication tool, The Visual Dump Analyzer will be installed on a shared Windows server. Then, to run The Visual Dump Analyzer you only need to create a normal Windows shortcut to VDA.EXE on the shared server, place it on your desktop and finally click on it. That is, nothing needs to be installed on your PC. You may choose to install The Visual Dump Analyzer locally on your PC, however, by simply copying the shared installation (probably in a directory called VDA or VDA50 including all subdirectories) to your PC, you can then execute VDA from that location.

If TM2200 is the selected communication tool, the Bea Tuxedo Workstation Client DLLs v 6.0 (or later) needs to be accessible to your PC. This is accomplished by adding a /TUXEDO subdirectory to the shared VDA installation on the Windows server and placing the Tuxedo DLLs within it. Once this is done, all VDA clients will automatically detect and use the Bea Tuxedo Workstation Client DLLs to access TM2200 on the mainframe. Thus, you do not have to perform any special Tuxedo installations on your PC for VDA to use TM2200.

Getting started – Auto Config

In this dialog, you will first need to decide which method of communication – TM2200 or TMVDA – you will use for accessing the Unisys mainframes. With either method, you will only need to enter the IP-address and the port number of the main development mainframe along with the listener port number of TM2200 or TMVDA.

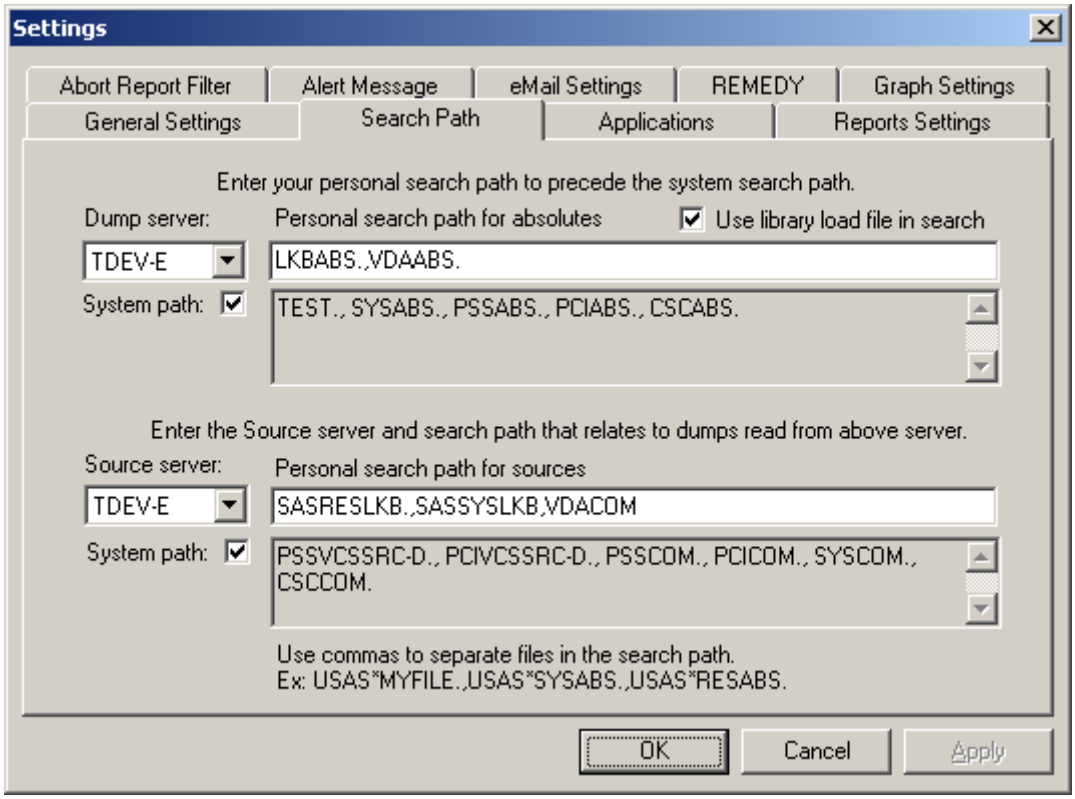


If you do not know the IP address of the main development server, just enter its name in the IP box as The Visual Dump Analyzer may know it and can thus set up the needed configuration using the name alone.

Once you have entered this information, press "**Auto Configure Now**". This will load the configuration for ALL of the Unisys servers, thus enabling you to access all of them.

Search Path setup

Before reading dumps, you must first setup your personal search path for both absolute and source elements. This will enable The Visual Dump Analyzer to examine the correct files when looking for absolutes and sources.



To update the search path, select "**Tools | Settings**" and then click the "**Search Path**" tab. In this dialog you can specify which personal files are to be searched before the files in the *system* search path. Be sure that you update the search path for all mainframes you access.

Although there is a *default* Source server for each Dump server that is defined within the configuration of The Visual Dump Analyzer, this can be overridden by selecting another server within the *Source server* drop-down menu when the related *Dump server* is displayed in its drop-down menu.

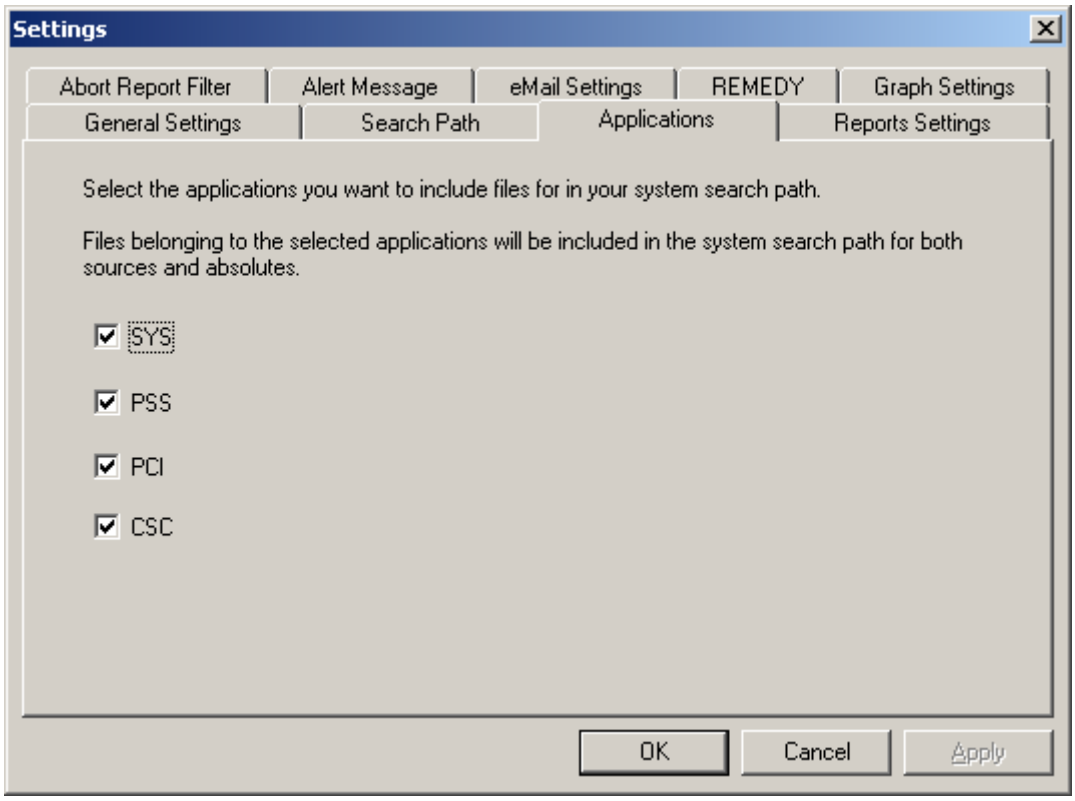
When searching for an absolute element, The Visual Data Analyzer will first search the file from which an absolute was loaded into the HVTIP library. If an exact match is not made with the absolute's creation date/time, your personal search path will next be examined, and finally the system search path will be scanned. If an exact match cannot be found, the absolute which *best* matches the date/time will be used. Note, if you do not wish to search the library load file, deselect the "**Use library load file in search**" option in this dialog box.

When searching for source code related to the dump file, The Visual Dump Analyzer will first examine your personal search path and then the system search path. Further, the date/time of the source code that is closest to the date/time of its related absolute element will be selected for the analysis. Note that unlike searching for an absolute element that can have an *exact* date/time match, looking for related source code will always require scanning all files in the search paths.

Absolute and source elements that are delete-marked within the files being searched will also be examined so, at times, it may be advantageous to not remove (i.e., pack) outdated versions of these elements. Also, if you

wish to not search the System search path for either absolutes or sources, deselect the path at the left-side of files listed within it.

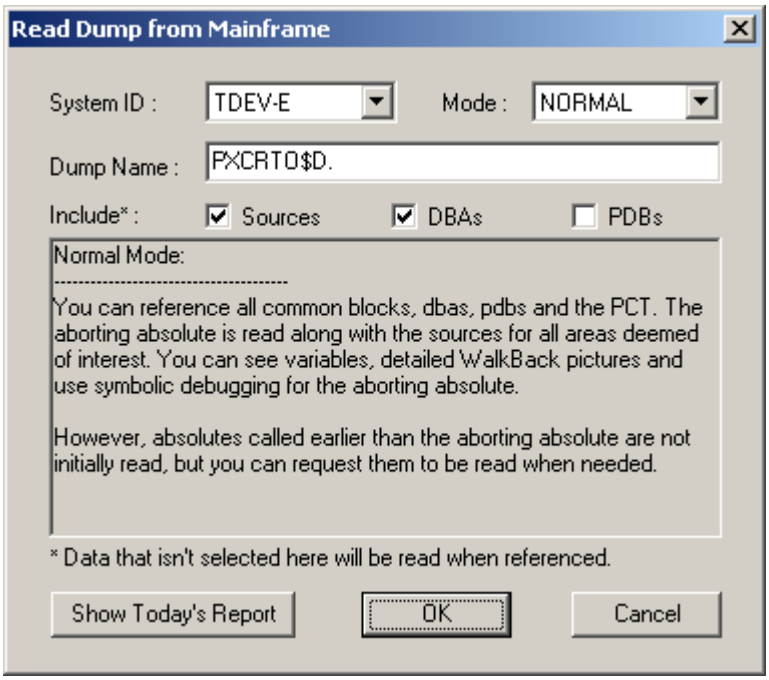
The system search path can be optimized by selecting "**Tools | Settings**" and then click on the "**Applications**" tab. The USAS applications listed here will reflect all applications that are included in the system search paths for absolute elements and source code. By deselecting an application in this list, files within the system search path that are associated with it (i.e., files having the same first three characters as the application) will not be traversed when searching for both absolutes and source code.



By deselecting applications that are known to not relate to the aborts that are being analyzed, considerable time will be saved whenever the table of contents for these files needs to be scanned.

Reading dumps from the mainframe

To read a new dump from the mainframe, select "**File | Analyze New Dump**" or click the "**Analyze New Dump**" icon. The following dialog will be displayed allowing you to specify the dump you wish to read.



Select the Mode (See ***Dump Analysis Modes***), click "**OK**" and The Visual Dump Analyzer will begin reading and analyzing the dump. This process will not be instantaneous because considerable information needs to be retrieved and correlated. Factors that affect the speed of this process include the Dump Retrieval Mode, the complexity of the dump, the number of Include: options checked, the power of the mainframe, the communication bandwidth and the speed of your PC.

To assist you in selecting the Dump Retrieval Mode most appropriate for a specific dump, the text window describes the possibilities you will have once the dump has been read and analyzed using the selected options.

The Visual Dump Analyzer obtains all the information it expects you will need for analysis during this process so you will receive quick responses during the time of examining the dump. If you have unchecked some of the *Include:* options and later discover that you need the information, it will automatically be retrieved when you reference it.

This dialog box also provides the option of viewing all the aborts that have occurred on a particular system today. By clicking on "**Show Today's Report**", the Abort Report will be retrieved and shown – providing you the ability of browsing the aborts and selecting the specific dump you want to read.

Dump Analysis Modes

Dumps can be downloaded and analyzed using three different modes. Each mode varies in the amount of assistance it provides by adjusting the depth of analysis performed by The Visual Dump Analyzer. This flexibility allows you to have the dump analyzed based upon your needs while still having the possibility of performing further analysis after the dump is initially retrieved.

Quick Mode

Being the simplest of the dump analysis modes, Quick Mode should be used when you know why the program aborted and thus wish to spend no time finding and analyzing the absolute and sources related to the dump. In

this mode you will have access to all D-bank data such as Common Blocks, DBAs, PDBs and the PCT. You will not, however, be able to view source code, see the values of variables, or Walk the Jump Stack.

The **WalkBack** picture will only show the flow between absolutes with no breakdown of the individual absolutes being performed.

Be aware that even though you cannot use symbolic debugging or see variables in Quick Mode, you can use the **Expression Window** to symbolically (by name) enter any Common Block define or main stack variable and have it resolved to its value.

Batch programs cannot be analyzed using Quick Mode because they each have unique mappings and thus require the absolute to be read and analyzed so the D-Bank structures can be accessed. If a batch dump is analyzed in Quick Mode, the mode will automatically be changed to Normal Mode once The Visual Dump Analyzer recognizes the dump is originated by a batch program.

Normal Mode

In Normal Mode you will have access to the same D-Bank data as with Quick Mode. In addition, The Visual Dump Analyzer will perform a complete analysis of the dump as it relates to the aborting absolute, Absolutes that are called earlier in the Program Call stack, however, will not be analyzed.

Besides having access to DBAs, PDBs, Common Blocks and the PCT; you can see the source code and use symbolic debugging. The aborting absolute's program stacks will be analyzed and presented in the **Variable Window.** Further, you have the possibility to **Walk the jump stack** in this mode.

The **WalkBack** picture will show the aborting absolute broken down into its relocatables and internal subroutines to give a detailed picture of the **WalkBack** within the aborting absolute. Absolutes called earlier in the program stack will not contain this detail.

While you are inspecting a dump read in Normal Mode, if you should request data from an absolute other than the aborting absolute, The Visual Dump Analyzer will recognize that this data is missing. In this case, The Visual Dump Analyzer will prompt you whether the missing absolute should be read and analyzed now. If you respond affirmatively, the missing absolute will be read and analyzed with the result being merged into the overall analysis of the dump. (See ***Reading or Replacing Absolutes.***)

By using Normal Mode, a dump can be first be analyzed rather quickly with much of the pertinent data being available. Then, if additional data is required from other absolutes, they can easily be integrated into the overall analysis. For this reason and because Normal Mode gives access to all the functions within The Visual Dump Analyzer, it is the default mode that is recommended for inspecting most dumps.

Full Mode

Full Mode is similar to Normal Mode except that ALL absolutes in the program call stack will be retrieved and analyzed during initial processing of the dump. In this mode, you will have access to all areas of the dump, be able to use symbolic debugging, and be able to see variables from all absolutes.

In Full Mode the **WalkBack** picture will display the complete path taken to the aborting location based upon the Program Call stack, with all the absolutes being broken down into its relocatables and internal subroutines.

The primary purpose of Full Mode is to ensure that all data is initially analyzed and transferred to the client so the dump can be inspected even after the dump has been removed from the mainframe. Therefore, use Full Mode whenever you want to save the dump to disc for later inspection or when you wish to e-mail the dump to someone for assistance.

Because a dump can potentially have many absolutes that relate to it, initially analyzing a dump in Full Mode can be more time-consuming. Once retrieved by the client, however, this mode is the most efficient.

Finding the Absolutes and Source Code

In order to use Symbolic Debugging, The Visual Dump Analyzer must have access to the correct versions of the program absolutes that created the dump. Generally, this is not a problem because the absolutes will be located in a system file for online dumps and in your personal file when developing code. The practice of leaving program absolutes in *temporary* mainframe files should be avoided because The Visual Dump Analyzer is not able to access them.

When The Visual Dump Analyzer cannot be certain whether it has found the correct program absolutes, you will be warned of the circumstance and be allowed to determine yourself whether the versions are actually correct.

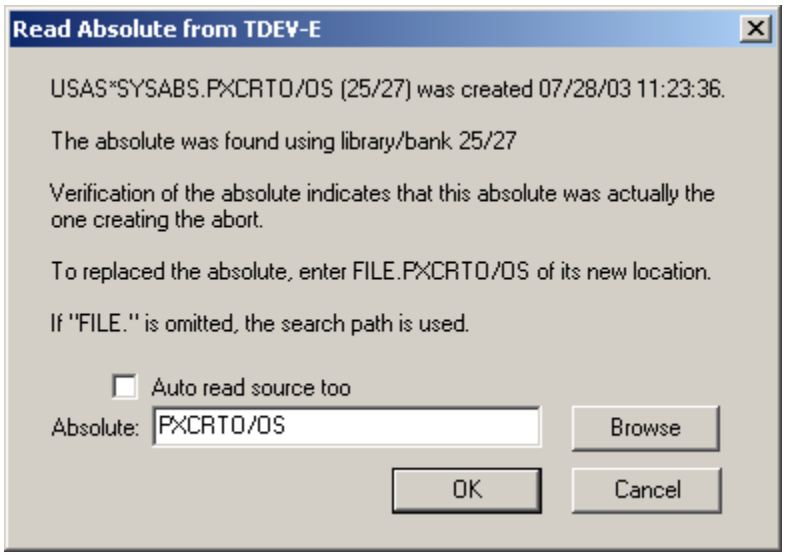
If the correct version of the program absolute is not found, you can still analyze global data areas within the dump such as DBAs, PDBs, and Common Blocks. Only the variables within the local storage (stack) of the programs related to the dump cannot be accessed. If a correct version of the absolute does exist but was not actually found, it can be easily be manually retrieved. (See ***Reading or Replacing Absolutes***.)

When the correct absolutes are found, they will be used to determine which source programs are to be read. Next, The Visual Dump Analyzer will check that the sources have not been changed since they were compiled and mapped into the absolute. Finally, it will verify that known line number references from the absolute actually match the source.

The result of the validity check performed on both the absolutes and the sources will be displayed using a single validity color code. This color code indicates whether the absolute matches the dump and, further, how well the sources match this absolute. The validity color code is shown as the vertical colored bar placed along the left border of the Source Display window. If this bar is not green, either the absolute or the source should be replaced. (See ***Validity Color Codes***.)

Reading or Replacing Absolutes

It is important to know whether the program absolutes correctly relate to the dump you are analyzing, otherwise it can result in misleading information. The absolute version can be checked by selecting the absolute and clicking the "**Absolute:**" button on the "**Source selection**" toolbar.



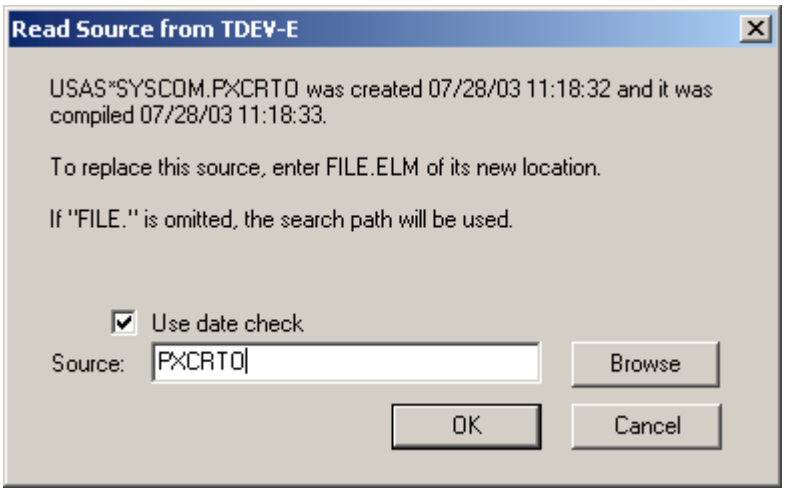
This dialog box will inform you of the location and version of the absolute that has been found, and allow you to replace it if you wish. Further, if the dump was initially retrieved in Normal Mode, only the aborting absolute is initially read and analyzed. By using this dialog box, you can request The Visual Dump Analyzer to read and analyze additional absolutes thus adding their analysis to the overall analysis of the dump.

Use the "**Browse**" button to scan the mainframe's program files to look for a different version of an absolute.

Reading or Replacing Sources

It is also important that you are using the correct source code version when you are analyzing a dump. You can always check the version of the source by clicking the "**Relocatable:**" button on the "**Source selection**" toolbar. This will display the below dialog box describing the details of the source and allow you to select a different source from the mainframe if it is incorrect.

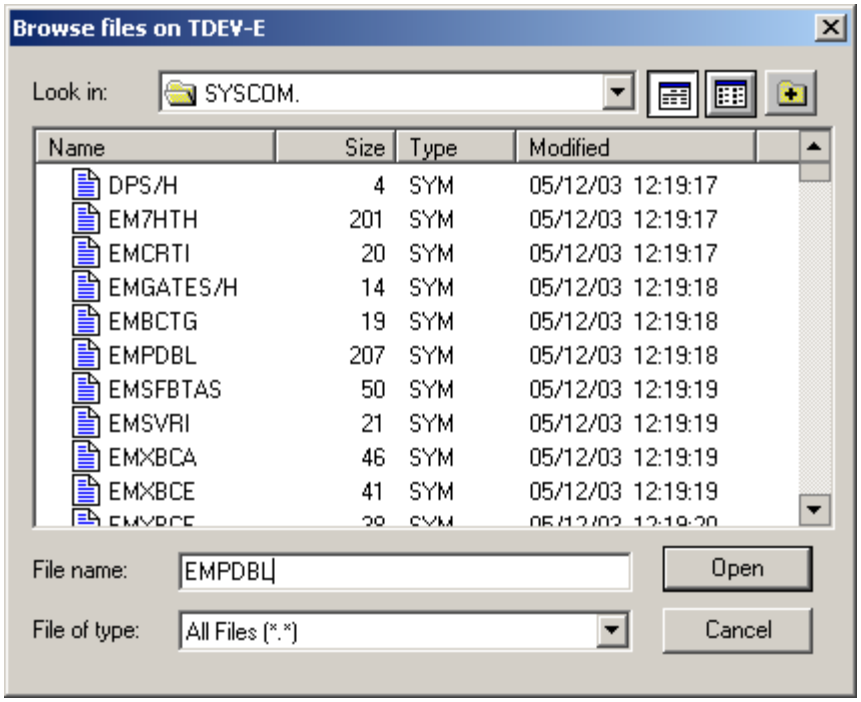
The below dialog box will allow you to read more sources than are initially read or to replace source code if it is determined to be incorrect.



Use the **Browse** button to scan the mainframe's program files to look for a different source code element.

Browsing Program Files for Absolute or Sources

The Visual Dump Analyzer enables you to browse U2200 files directly on your PC. Use this method to locate the correct version of an absolute or source code.



When browsing for source code on the mainframe, the "**Look in:**" drop-down box will contain all the files defined in your source search path. If you wish to browse a different file, you can add more files to the "**Look in:**" drop-down by clicking the add (+) button.

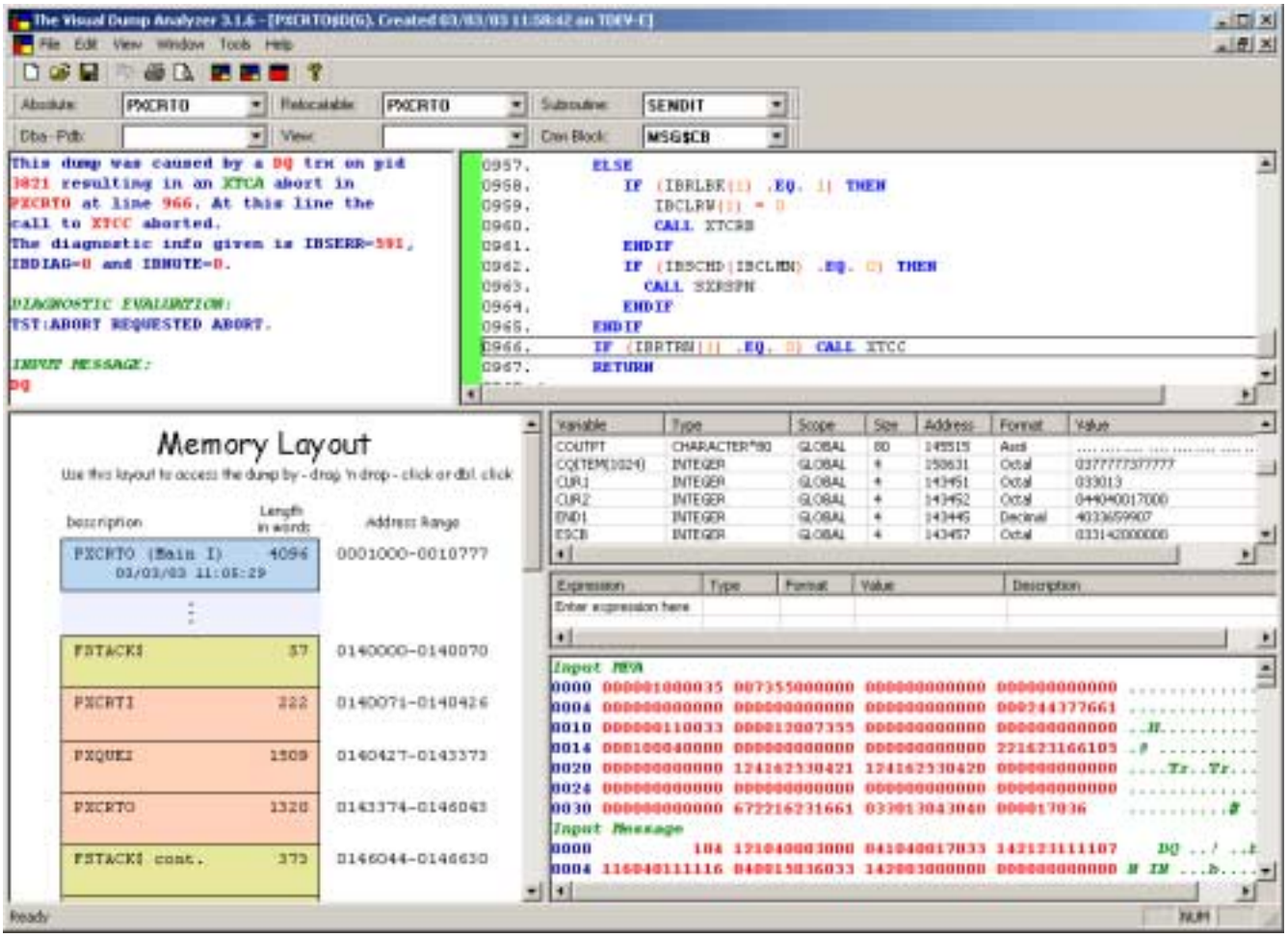
The contents of the program files in this dialog box are retrieved from the mainframe at the time the file is selected. Thus, the elements that are shown are always current.

When browsing for absolutes, the "**Look in:**" drop-down box will contain all the files defined in your absolute search path.

You can select a source or an absolute in the browse dialog by double-clicking it. This will return you to the original "**Read Source**" or "**Read Absolute**" dialog with the selected source or absolute being presented.

Validity color codes

When the dump is read and the source code is displayed in the **Source Display** window, its validity is shown using color codes in the vertical status bar (shown here in **GREEN**).



The color codes indicate the following:

GREEN:	You can use Symbolic Debugging (i.e., you can trust the value of all local variables) since both the absolute and the source elements are valid.
YELLOW:	You can use Symbolic Debugging because the absolute is valid, but you cannot rely on any line number references since the source is not valid.
GRAY:	The validity of the absolute cannot be determined, thus no check can be performed on the sources either. Use Symbolic Debugging only if you know that the absolute and the source are valid.
RED:	The absolute is not valid. Do not use Symbolic Debugging. Trust GLOBAL areas such as Common Blocks, DBAs and PDBs only.

To receive a description of these color codes, simply click anywhere on the status bar in the **Source Display** window.

Saving dumps to disc

Once a dump has been read from the mainframe, you can save it to a local disc so that it can later be retrieved. This is often a good idea if you wish to continue examining the dump at a later time, or wish to save the dump for archival purposes.

To save the dump to disc, just select "**File | Save Dump to Disc**" or click the "**Diskette**" icon and the normal Windows dialog for saving files will be displayed.

The Visual Dump Analyzer saves the file in its native ascii format. It will contain all information needed to reestablish the environment that you are in including the dump, its analysis, and the related absolute and source elements.

Storing dumps on your local disc has benefits. Even when the dump has been removed from the mainframe or any of the related absolutes or sources are deleted, you still have the capability of examining the dump. Furthermore, retrieving the dump from a local disc is considerably faster than reading it from the mainframe, so if you expect to open it again for analysis, saving it to a local disc will save time.

When you wish to save the dump to disc so you can later inspect it when the dump may no longer exist on the mainframe, be sure to retrieve ALL the needed information before saving the dump. (See ***Emailing dumps***).

Opening dumps saved on disc

You can retrieve dumps that are saved locally by selecting "**File | Open Saved Dump or Report**" or clicking the "**Open Folder**" icon. This will display the normal Windows dialog for opening files. All information related to a dump is kept in the disc file so you won't lose any information or functionality by saving a dump to disc and then rereading it.

The Visual Dump Analyzer supports a multi-document environment, thus you can have multiple dumps or Abort Reports opened at the same time. Use the **Window** command in the main menu to arrange and move between open dumps.

Emailing dumps

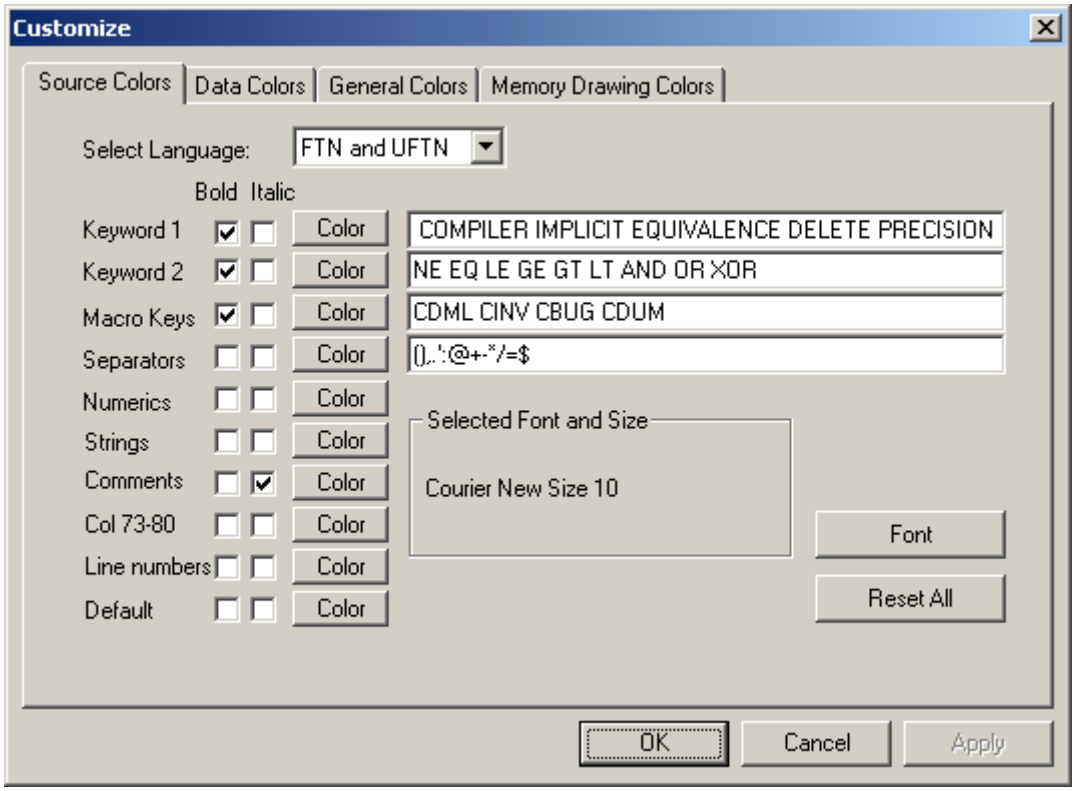
Once a dump has been saved to local disc, it can be attached to an email just like any other file. Alternatively, you can click the mail icon on the main menu and your configured mail client will be opened with the dump being attached to the new e-mail that is prepared to be sent. Note that dumps can be sent to different sites with the recipient being able to view the dump as long they have the Visual Dump Analyzer client installed.

Before sending a dump, you should consider whether the recipient has access to the USAS dump$d file on their mainframe at the time they are analyzing it. If they may not and because only the information retrieved from the server is saved in the dump, you should retrieve all the data, absolutes and source code that may be needed by the recipient. Because this may be difficult to know, it is safest to include as much as possible.

Preferably, you should use **Full Mode**, or at least ensure that all absolutes are read if the dump was originally retrieved in **Normal Mode**. You should also consider if the PDBs need to be included as part of the dump retrieval. And finally, you may want to retrieve the IB$F and the CA$F views along with any specific application views that could be of interest as this will enable these data blocks or DBAs to be referenced using their symbolic names. This can be important because if the dump is shipped to a different site, the location of individual CA and IB block fields may not be the same.

Customizing colors and fonts

You can change the colors and fonts that The Visual Dump Analyzer uses by selecting "**Tools | Customize**". Different colors and fonts can be set for each of the main window types by selecting the tabs at the top of the dialog box. Upon clicking "**OK**" on this dialog box, the changes will have immediate affect. To revert back to the default settings, click "**Reset All**".

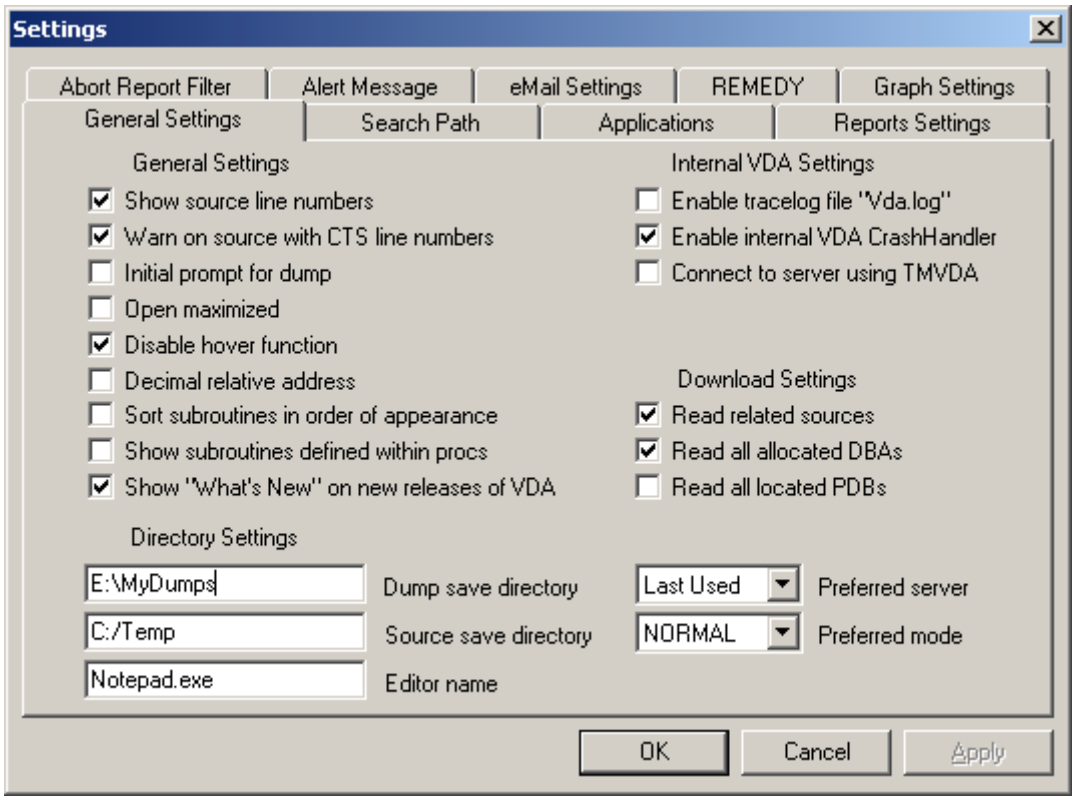


Customizing settings

You can use "**Tools | Settings**" to customize settings within The Visual Dump Analyzer.

You should create a directory on your PC that will be used as the default directory when saving and reading dumps. Specify this directory in the **"Dump save directory**" prompt.

You should also create a directory on your PC that will be used to save program sources. This directory will be used when you request The Visual Dump Analyzer to open a source in your preferred editor. Specify this directory in the "**Source save directory**" prompt.



Viewing the dump

The Visual Dump Analyzer uses splitter windows of six different types in displaying the information contained within the dump. Further, each window type has its own Pop-up menu that allows you to use the functions that it supports. Some of these functions are accessible from all window types while others are specific to an individual one. To display its Pop-up menu, simply right-click within any of these windows.

The **Source selection** and **Data selection** toolbars control the contents of the splitter windows. These toolbars are initially docked at the top border of the main window although you can drag them to any other border or have them floating on the screen.

Resizing the windows

You can use the three-resize buttons (those which look like the VDA logo) in the main menu to change the size of the windows. The left-most resize button will set the windows to be of **equal** size. The middle resize button will set the windows to their **default** sizes. And the right-most button will **maximize** the active window.

You can also quickly maximize a window's size by double clicking on it. Then, if you double-click on it again, the windows will return to their original sizes.

For more control in sizing windows, you can also drag the borders between windows.

Setting the active window

The *active* window is the window that has most recently been clicked in. When applicable, the entries within the toolbars describe the contents of the active window.

Splitting the windows

The **Split window** function is common to the Pop-up menu for all the window types (except the Expression Window). This function allows you to horizontally split one window into two and then work with the second one independent from all others. The new window, which will initially take on the characteristics of the window it has been split from, can later be removed by dragging its border all the way to its top or bottom.

There is a limit of one split window for each window initially displayed by The Visual Dump Analyzer.

Changing window types

The **Change window to** function is also common to the Pop-up menu for all window types (except the Expression Window). This function allows you to change the view of a window into another window type. For example, after you have read the text in the **General Info** window, you may decide that you no longer need it and would rather have another **Data Display** window. By selecting the "**Change window to | Data Display**" function, you can easily do this.

The Source selection toolbar

This toolbar contains three buttons with an associated drop-down list for each. Simply press the *text* of the corresponding button to obtain information related to the selected absolute, relocatable, or subroutine.

Click on the "**Absolute:**" button to view the creation date/time of the selected absolute, its validity related to this dump, the mainframe file it was found in, and the method that was used to find it.

Click on the "**Relocatable:**" button to view the compilation date/time of the selected relocatable. When the source code found by The Visual Dump Analyzer is not the one you desire, use this button to enter a dialog that allows you to browse the mainframe files to select a new one.

Click on the "**Subroutine:**" button to view the date/time the selected subroutine was created along with the line numbers it spans within its element.

You can use this toolbar to change the contents of an *active* **Source Code** window by selecting a different subroutine or relocatable within the drop-down lists. If multiple **Source Code** windows are in view, the selected entry within the drop-down list indicates the one that is currently (or most recently) active. Moreover, if you wish to change the contents of a **Source Code** window, just click on it to make it active and then select a different entry within these drop-down lists.

The Data selection toolbar

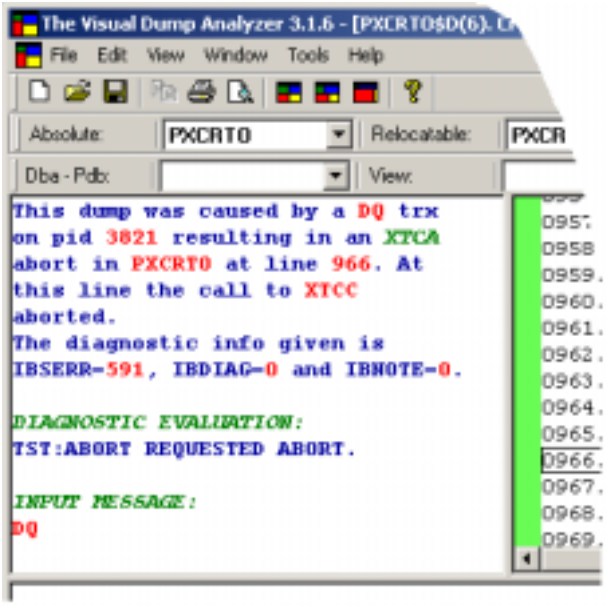
This toolbar also contains three buttons with an associated drop-down list for each.

Click on the "**Dba - Pdb:**" button to view information on the selected data area.

Click on the **View:** button to select a private view or a view from a specific file (See ***View Construction***). This dialog will allow you to browse through your files on the mainframe searching for a specific view. All public *views* are always in the drop down-list for easy selection.

Click on the **Cmn Block:** button to view information on the selected common block, including its location.

You can use this toolbar to change the contents of the *active* **Data Display** window by selecting a different DBA/PDB, view or common block from the drop-down lists. If multiple **Data Display** windows are displayed, the selected entry within the drop-down list indicates the one that is currently (or most recently) active. Moreover, if you wish to change the contents of a **Data Display** window, just click on it to make it active and then select a different entry within these drop-down lists.



The General Info window

This window provides you with a natural language explanation of what has caused the dump.

The explanation provides all relevant XTCA and CONTINGENCY (including HVTIP) error codes along with a complete description of these codes, when applicable.

This window also contains I/O error information by gathering all the FCSS diagnostics, translating it then to natural language and including it in the evaluation.

Similarly, EDIFACT errors returned from either the edifact input or output handler are translated and included in the evaluation.

If any hints are contained within The Visual Dump Analyzer's **Knowledge Database**, they will also be included here. (See ***The VDA Knowledge Database***)

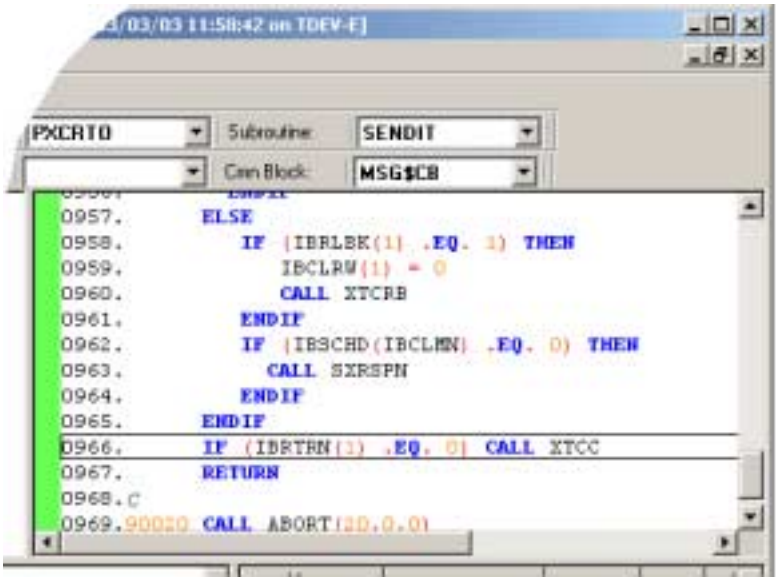
When you have solved a dump, you can select "**Update Hint**" from this window's Pop-up menu and save the solution in the VDA Knowledge Database. Then, the next time a dump similar to this one is being analyzed, your notes will automatically be displayed in this window.

The Source Code window

This window displays the source code that is applicable to the dump. You can change the contents of this window by selecting a different entry within the "**Subroutine**" drop-down menu of the **Source Selection** toolbar or by dragging a source code block from the drawing window. (See ***The Drawing Window*** section)

The left side of this window has a color-coded status bar indicating whether the absolute and displayed source code is valid for this dump. (See ***Validity color codes***) You can click on this status bar to receive a description of the validity status.

If this bar is **GREEN**, you are free to use the complete set of Symbolic Debugging functions within this window. If the bar is not, you are responsible for determining whether the absolute and source code relates to the dump that you are viewing.



Symbolic Debugging

When the status bar is **GREEN** or **YELLOW**, you can click on any variable, define, or parameter within the source code and its value will be displayed in a small box adjacent to the tag you click on. This will quickly provide you with many answers related to the status of processing shortly before the dump occurred. These values are presented in octal, decimal, ascii, and fieldata.

Pop-up Menu Functions

There are many helpful functions that you can perform via the Pop-up menu of the **Source Code** window. A description of these follows:

When the status bar is **GREEN**, you can **Walk the program call stack (WalkBack)**. Here you can *symbolically* follow each forward call taken by the processing. Walk this path forwards or backwards routine-by-routine from the very first call from the ICP until you end in the aborting subroutine. (See ***The Program Call Stack (WalkBack)***).

When the status bar is **GREEN**, you can **Walk the jump stack** for all contingency type aborts. Here you will be able to view the contents of the entire Jump Stack *symbolically* – never having to look at a single octal address. After selecting this option, step-by-step you will be moved between the lines of code that have executed just prior to the abort occurring.

"**Go to Last ACB call**", "**Go to Abort address**", "**Go to FCSS I/O error**", and "**Go to X11 return address**" all provide quick ways of going directly to the source code line where an important event occurred related to this dump. Again, you will be able to view this location symbolically without having to do any address manipulation.

Go to Address and **Go to Line Number** are quick ways for you to view source code at a particular location.

"**Return to Caller**" and "**Go to Forward Call**" allow you to move one step backwards or forwards in the "**Program Call Stack (WalkBack)**" from the present location - enabling you to return to the calling routine or move forward to the next forward call. (See ***The Program Call Stack (WalkBack)*****).**

"**Show MASM code**" lets you view the actual MASM code that is generated by the compiler for the source code you are viewing. By selecting this option, the window will expand into having each source line displayed with the

MASM code it produces listed just below. When using this function, be aware that all sources lines do not produce MASM instructions. Further, depending upon the compiler options used, the MASM instructions may be shifted up/down and overlap more sources lines in order to optimize performance.

"**Obtain more MASM code**" should be selected when the MASM code that is initially shown does not cover all the source lines you are interested in. To save time, only the MASM code *around* the source code you are currently viewing is retrieved. You are free to obtain more MASM code by selecting this function.

"**Open Source in Editor**" lets you view the source code within an editor of your choice. That is, you can use the full features of an editor while still having The Visual Dump Analyzer active. To configure the editor you wish to use, select "**Tools | Settings**" and enter it in the "**Editor name**" prompt.

The Variable Window

You can symbolically view local and global variables by using the **Variable** window. This window will always display variables that are associated with the code being viewed in the **Source Code** window. Moreover, the register set can also be viewed in this window.

You can arrange the columns in this window by simply dragging their header to its new location or sort the columns by clicking their header.

Variable	Type	Scope	Size	Address	Format	Value
CQITEM(1024)	INTEGER	GLOBAL	4	150631	Octal	0377777377777
CUR1	INTEGER	GLOBAL	4	143451	Octal	033013
CUR2	INTEGER	GLOBAL	4	143452	Octal	044040017000
END1	INTEGER	GLOBAL	4	143445	Decimal	4033659907
ESCB	INTEGER	GLOBAL	4	143457	Octal	033142000000
ESCM	INTEGER	GLOBAL	4	143460	Octal	033115
ESCM1	INTEGER	GLOBAL	4	143447	Octal	033115000
ESCM2	INTEGER	GLOBAL	4	143450	Octal	033115000000
FCC1	INTEGER	GLOBAL	4	143431	Binary	11111000100100000100000
FCC2	INTEGER	GLOBAL	4	143432	Binary	11110000011000000000000000000...
IBBUFF(120,1)	INTEGER	GLOBAL	4	155301	Decimal	402917376
IOPMSG(7)	INTEGER	GLOBAL	4	145506	Octal	052052052052
IOPTXT	CHARACTER*28	GLOBAL	28	145506	Ascii	**** OUTP UT I NHIB ITED ****
IOUTPT(20)	INTEGER	GLOBAL	4	145515	Decimal	0
LEVEL	INTEGER	GLOBAL	4	143401	Real	1.423194e-045
LWC	INTEGER	GLOBAL	4	143411	Binary	10000000100110000010 1
LWCO	INTEGER	GLOBAL	4	143461	Decimal	0

Clicking on some of the cells within the table will result in the contents being adjusted. If, for example, you click on a variable that is an array, the array will be expanded with all its values being shown. Clicking on it again will reduce the variable to one entry in the table. If you click on a cell within the Format column, the value of the variable will cycle through a wide range of data types including decimal, octal, ascii, fieldata, real, date, binary and logical. Finally, if you click the value of a variable, the value will be shown in a small box adjacent to the cell. Its value will be displayed in octal, decimal, ascii and fieldata

By right-clicking and displaying the Pop-up menu of this window, you can select/deselect the type of variables displayed within this window. Note that static variables here refer to variables defined within the static areas of the d-bank. In this Pop-up menu, you can also choose to display the AXR$ registers.

The Expression Window

The **Expression** window allows you to request the value of an *expression*. The expression you enter can be as simple as the name of a define, constant, or variable – or it can be a complex mathematical expression containing any combination of these.

To determine the value of an expression, click on the first empty row and enter the expression you wish to have resolved. In the example below, IBDIAG(1) has entered as the expression on the first line.

Expression	Type	Format	Value	Description
12872*28	DEFINE	Decimal	360416	
CABCB(2)	DEFINE	Octal	07355	
IBDIAG(1)	DEFINE	Decimal	0	
IOPTXT	VARIABLE	Octal	052052052052 117125124120 125124040111 1161101...	
IOPTXT(5:6)	VARIABLE	Octal	117125	
Enter expression here				

After an ***expression*** is entered, The Visual Dump Analyzer calls a processor on the mainframe that will use the dump and active absolute in order to resolve it. The answer will then be returned and shown in the Value column.

There are few limitations with the complexity of the ***expression*** you may enter. It is only required that the process on the mainframe can understand all of the symbols you use within the ***expression***.

The mainframe process automatically understands many symbols. These include ones defined within SPSYS, SYS Common Blocks and variables defined within the active absolute. Therefore, symbols prefixed with S$xxxx, CAxxxx, IBxxxx, CBxxxx, CQxxxx, EYxxxx, TAxxxx, among others can all be used.

Beside these symbols, you can *teach* The Visual Dump Analyzer any application defines for PDBs or DBAs by a method that creates **Resolve Programs**. (See ***Resolve Programs*** section later in this manual)

The following are examples of ***expressions*** you can have resolved:

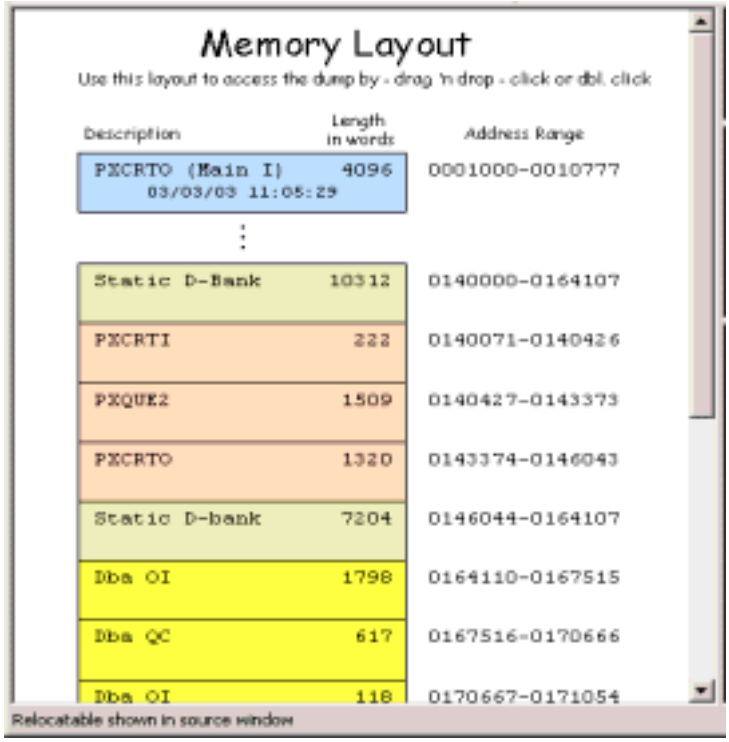
Expression	**Action**
MYTABL(X+2)	Resolve the value of the table MYTABL defined in the active absolute using the index X+2, where X is a variable in the active absolute.
MYVAR(X,10,Caxxxx(1))	Resolve the value from the 3-dimentional table MYVAR defined in the active absolute using the value of X from the active stack as the first index, 10 as hard-coded value for the second index and a CA-define as the third index.
MYSTR(IDX)(10:Y)	Resolve the value of the substring (10:Y) of the string MYSTR defined in the active absolute by using IDX as the index and Y as the end character position. Both IDX and Y are assumed to be variables in the active absolute.
MYVAR1+3/(3*MYVAR2)	Resolve the value of the mathematical expression using the integer variables MYVAR1 and MYVAR2 from the active absolute.
FIATYP	Resolve the value of FIATYP variable from the FI DMS record.
IBDIAG(1)	Resolve the value of the SYS define IBDIAG using a hard-coded index of 1.
MYDEF	Resolve the value of the application define MYDEF using the DBA from the dump and a pre-produced resolve program.
12875*027	Resolve this mathematical expression – assuming 12875 to be decimal and 027 to be octal.

The "**Show View values**" option within the Pop-up menu for this window allows you to see all the expression names resolved during the analysis of this dump.

The Drawing Window

This window provides you with a graphical overview of memory at the time the dump occurred. This comprehensive picture of the memory layout includes the instruction banks, static and dynamic data banks, the Public Data Bank (PDB), the Program Control Table (PCT), the Extended Mode Activity Local Stack (ALS), the Return Control Stack (RCS), and any HEAP or POOL banks allocated by the transaction.

You can view each main area as a single composite or – by simply double-clicking on it – break it down into its basic components. That is, the instruction bank can be decomposed into all its relocatables; the static data bank can be decomposed into its many common blocks; the D$RDA can be decomposed into its many DMS records, and the dynamic data bank can be decomposed into its allocated DBAs, free areas, and program stacks.



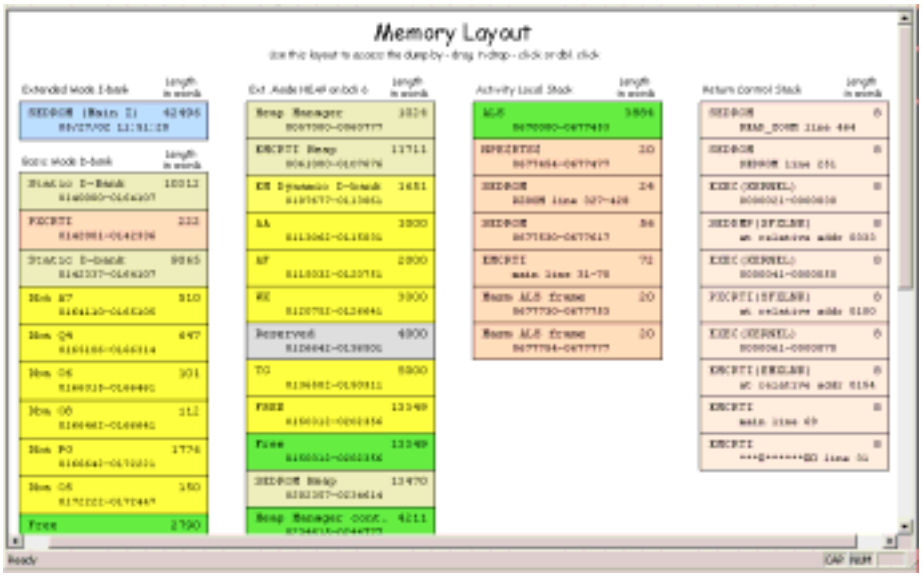
When a data area is overlaid by the following one, dashed lines are drawn to separate the two blocks and the address range is only shown for the first block. This is commonly seen with DMS records and static DBAs.

By using the Pop-up menu associated with this window, you can toggle whether an area will be included in the drawing or whether is should be decomposed into its specific parts.

The drawing can be viewed in a Fixed Layout where all blocks are the same size, Proportional Layout where block sizes are related to the actual memory size, or Adjusted Layout where a combination of these is used. This feature is also adjusted with the Pop-up menu.

The **Drawing** window uses color-codes for each memory area so it can be easily recognized. Moreover, errors that exist within memory such as FCSS errors or Corrupt Free Chain will be color-coded **RED** so they can be quickly perceived within the drawing.

You can also use the **Drawing** window as an *access pad* for changing the contents of one of the other windows. By simply *dragging* an instruction or data block from this drawing and then *dropping* it into another window, you will be able to view its contents within a **Source Code** or **Data Display** window. This drag-and-drop feature is handy and addictive.



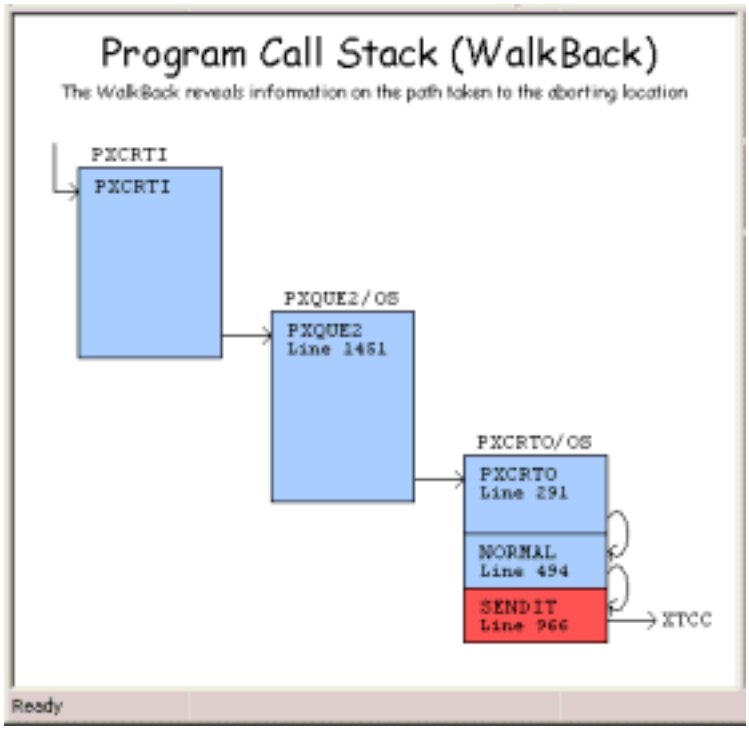
If you wish to quickly view the BCB in a DBA or the contents at the beginning or the end of a memory area, you can click the start/end address shown, and the first/last six octal words will be displayed in a small Pop-up box.

In general, the **Drawing** window provides an ideal way of viewing memory at the time of an abort, and provides easy access for viewing the contents of each area.

The Program Call Stack (WalkBack)

The **WalkBack** is found in the Pop-up menu for the **Drawing** window. When selected the drawing will change from the Memory Layout to the **WalkBack** picture.

The information provided by the **WalkBack** picture reveals the path the processing took in getting to the abort location.



This picture contains the actual calling sequence of the programs that have program stack areas within the dump. That is, the location within the source code is shown where each program calls forward to an internal subroutine, a mapped in relocatable, an ACB routine or another HVTIP program.

Each of these calls is drawn as a separate box grouped into absolutes. Internal subroutines are drawn directly *appended* to the absolute whereas external mapped in subroutines are drawn *under* the calling absolute.

Each box will contain a short written text stating the location of where the forward call was made. For Fortran programs, the text is the source line number and for masm programs it is the absolute address.

The aborting subroutine is drawn in **RED** and the active subroutine – the one currently displayed in the **Source Code** window - is drawn in **GREEN**.

The **WalkBack** picture can also be used as an *access pad* for changing the contents of the **Source Code** windows. By simply *dragging* a subroutine from the picture and *dropping* it into another window, you will be able to view the source code about the line that produced the forward call. Note that you can drop the subroutine into any window type with the destination window automatically changing to a Source Code window.

The **WalkBack** picture starts from the first forward call within the dump file. Normally, this will be the call made by the ICP program calling an application program. If the transaction has made a program *transfer* during execution, however, the record of calls made before the *transfer* is erased and the **WalkBack** picture will start after the program *transfer*.

To enable the Visual Dump Analyzer to perform the **WalkBack** logic, it is essential that the correct absolute elements are found. If an incorrect absolute is used, the HVTIP program stack cannot be decomposed into all of its internal subroutines, thus forcing the HVTIP stack to be displayed as one single box.

Finally, this display should be considered an analysis of the program stack areas and not a detailed trace of the program's execution. That is, this path does not include *side trips* to subroutines that have already been returned from as it is not a *trace*, but instead is a *snapshot* of memory at the time of the abort.

If a dump has been read from the server using normal mode – The Visual Dump Analyzer may not have read all the absolutes during the initial analysis. The absolutes that are not yet read are not broken down and interrogated and are thus shown in the **WalkBack** picture without any detailed information. However, by double clicking one of these absolutes in the drawing you will inform request The Visual Dump Analyzer to read and interrogate the absolute and update the **WalkBack** drawing with detailed information on this absolute.

The Data Display Window

You can view raw data from the dump in a variety of *formats* by using the **Data Display** window. The formats are selected by using the Pop-up menu for this window type. To set your preferred data display format, just enter "**Customize | Data colors**" and select an entry within the "**Initial data format**:" drop-down menu.

The Visual Dump Analyzer is aware of many data areas that are divided into smaller sections. To provide a better overview of the data, these areas are automatically divided so each section is clearly labeled with its own title and relative addressing.

```
Input MPA
0000 000001000035 007355000000 000000000000 000000000000 ................
0004 000000000000 000000000000 000000000000 000244377661 ................
0010 000000110033 000012007355 000000000000 000000000000 ..H.............
0014 000100040000 000000000000 000000000000 221623166105 .@ ...........vE
0020 000000000000 124162530421 124162530420 000000000000 ....Tr..Tr......
0024 000000000000 000000000000 000000000000 000000000000 ................
0030 000000000000 672216231661 033013043040 000017036    ..........# ...
Input Message
0000          104 121040003000 041040017033 142123111107    DQ ..! ..bSIG
0004 116040111116 040015036033 142003000000 000000000000 N IN ...b.......
0010 000000000000 000000000000 000000000000 000000000000
```

An example of an area that is split into multiple sections is the CABLOC. It is divided into the CABCB, Screen Addendum, Agent Addendum, General Information Area, Application Table and Application Area.

A second example is the MSG$CB which is split into an Input MPA, Host-to-host header (if present), Input Message, and Output MPA.

Besides being able to change the display format of the data, you can select from several options within this window's Pop-up menu:

"**Show absolute address**" and "**Show relative address**" inserts/removes absolute and relative addresses in the data view.

"**Show BCB**" toggles the display of a six-word BCB when a DBA is displayed.

"**Break long lines**" forces all lines that are longer than the width of the window to be broken (insert CR/LF) and thereby ensure the entire line can always be seen. All data will be placed in aligned columns.

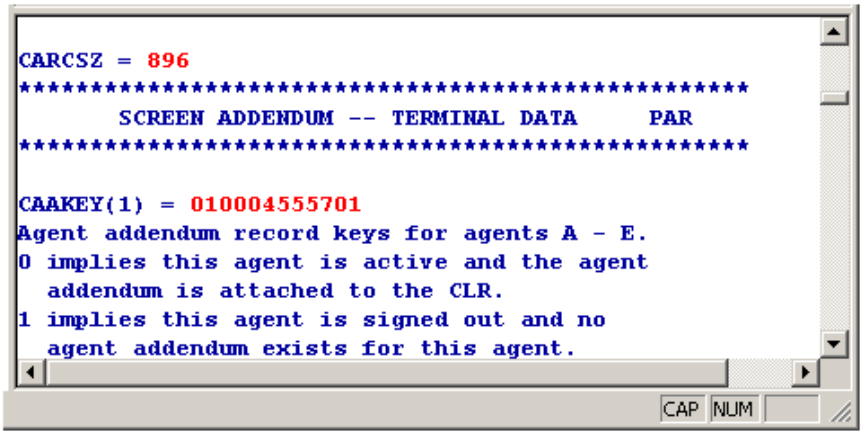
"**Display at Address**" lets you display data beginning at an address or entry point that you specify.

"**Display DMS record**" allows you to choose a DMS record from a selection list of DMS records present in the DMS Schema used by the active program. This allows you to access DMS records by name without knowing their internal DMS record number or location.

"**Display data as a View**" is a short cut method of displaying the block of data present in the window as a *view*. Essentially, it is the same as selecting a view in the **View:** drop-down menu of the Data selection toolbar.

"**Open Dba in editor**" lets you view the data within an editor of your choice. That is, you can use the full features of an editor while still having The Visual Dump Analyzer active. To configure the editor you wish to use, select "**Tools | Settings**" and enter it in the "**Editor name**" prompt.

"**Find…**" can be used to search for a string of characters within the window. The standard Windows function is used for this, thus providing you with various options in the search.



The **Data Display** window can also be used to present data that is overlaid with a *view*. *Views* provide a powerful and flexible way of presenting data by having the ability to incorporate freeform text and symbolic data from the dump resulting in a very helpful natural language description.

Views can be automatically generated from procs or be manually developed for specialized use.

When Views are automatically created based on a $F or –F proc, they will show the data as defined by the proc itself. All documentation text will also be preserved with the actual define statement only being replaced with the value of the field, thus generating views that look very similar to the proc themselves. This not only makes the procs easy to understand, but ensures the data being read uses the same defines as the program, which is essential when a dump is caused by flawed defines rather than corrupt data. (See ***View Construction***)

Rescheduling the Input Message

The Visual Dump Analyzer allows you to again reschedule the input message from a dump. There are many benefits with rescheduling input messages, all of them assisting you in finding and fixing the cause of an abort. The reasons for doing this include:

- Setting and running Traces
- Causing the dump to be taken at a more informative location
- Testing and verifying that your fix has worked
- Causing the same abort on a different system

Many of the input messages used in USAS today are Edifact messages and are thus not easily reproduced manually from an HVTIP screen. The Visual Dump Analyzer lets you re-enter these large messages without requiring them to be resent by another system.

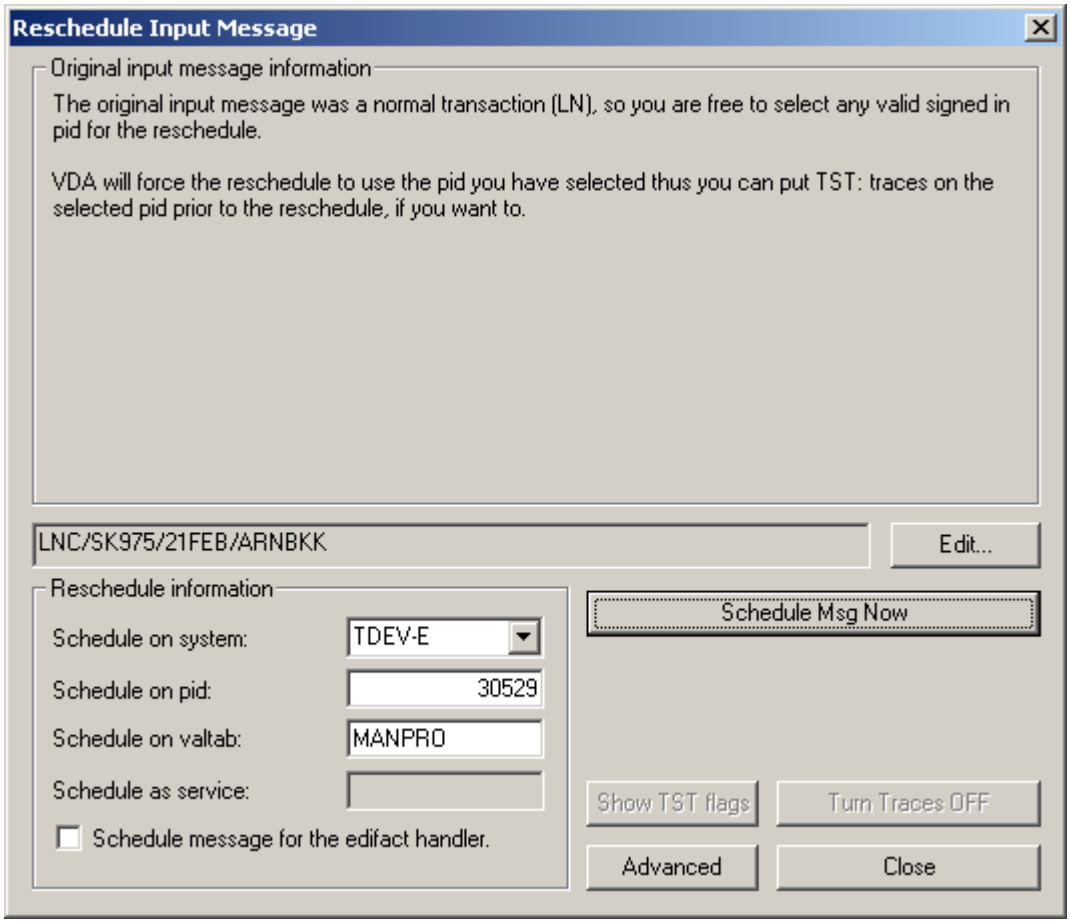
All aborts cannot be reproduced by simply rescheduling the input message again, but in cases where this is possible, it can be extremely helpful for interrogating an abort. Moreover, when an abort occurs on a production system, it can be very beneficial towards solving the abort when it can be reproduced on a test system. For this reason, The Visual Dump Analyzer allows you to reschedule the input message on a different system. For safety reasons, however, rescheduling messages on a production system is not allowed.

Dumps are often not taken at the most informative and beneficial location. Therefore, forcing a dump to be created at the right spot and at the right time is often the best way to solve a problem. This can often be very difficult to accomplish. By combining the Visual Breakpoint Traps logic with this rescheduling mechanism, however, it is greatly simplified. You only need to decide which line number along the path of the program logic that you would like the dump to be taken and then right-click upon it. This will set the Visual Breakpoint Trap. Once set, you can reschedule the message again, which will force a Breakpoint Abort – Contingency 014 – at the specified line number in the rescheduled transaction. Now you will have a new and more informative dump to study. (See ***Setting Breakpoint Traps***).

Finally, you can use the reschedule logic to test and verify that you have repaired the problem by again rescheduling the input message while executing the program that you have just fixed.

Rescheduling normal terminal transactions

To reschedule an input message, select "**Tools | Reschedule Msg…**" from the main menu or, alternatively, right-click in the **General Information Window** to open the Pop-up menu and select "**Reschedule Msg…**" to display the dialog below.



When rescheduling a normal VDU transaction, you can choose to reschedule the input message on another PID or another VALTAB. It is your responsibility to ensure that the selected PID is signed in and is allowed to run the transaction. Generally, a VALTAB should only be changed if you have added extensive traces to the transaction so that a reschedule using the normal VALTAB would cause it to timeout before the abort.

To reschedule the input message again, simply press "**Schedule Msg Now**" and the message will be rescheduled. Next, an information dialog box is displayed with information on the result of the reschedule. Finally, you can use TST:PRINT from a normal HVTIP screen to capture the trace. Naturally, if the reschedule causes a new abort, you can open the new dump by using The Visual Dump Analyzer.

Rescheduling HTH messages

Host to Host (HTH) messages are more difficult to reschedule than normal VDU transactions because the HTH header along with the HTH configuration determines which PID the transaction will execute on. As a result, the PID can often not be predicted making it impossible to turn on traces beforehand.

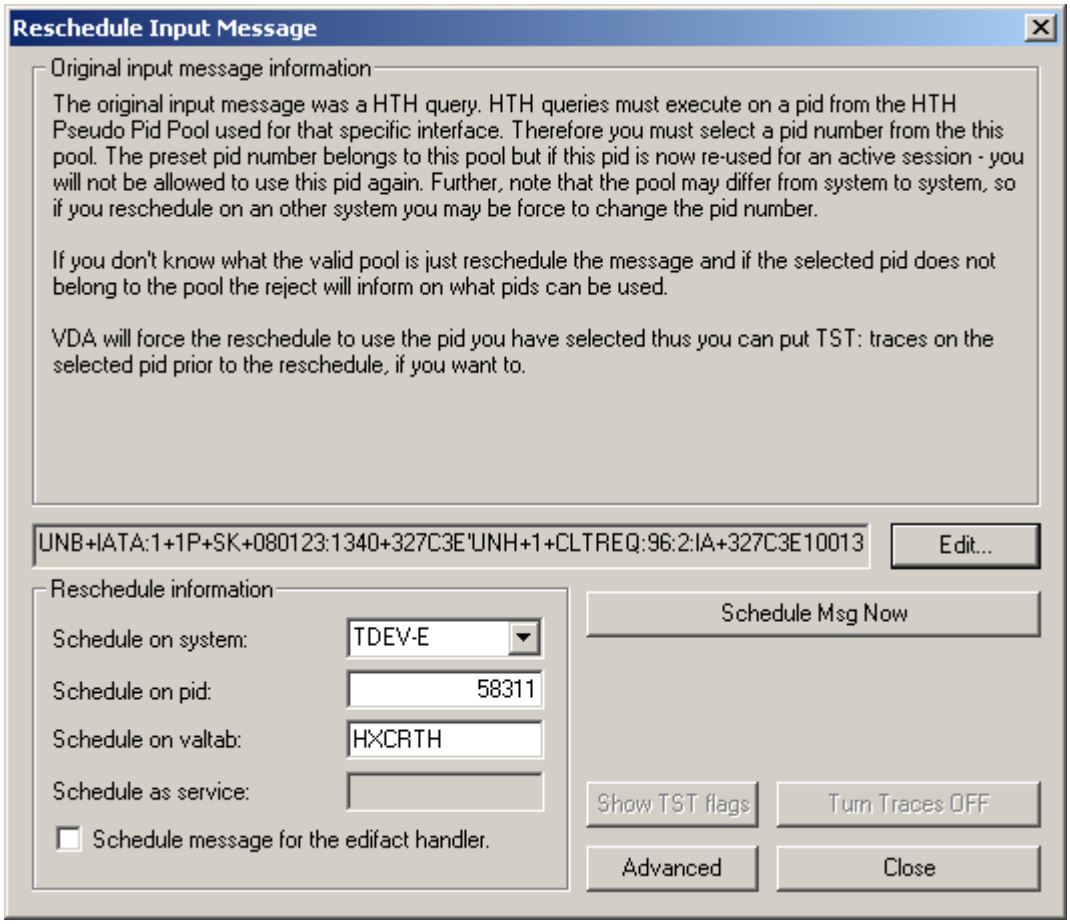
The Visual Dump Analyzer solves this problem by allowing you to choose the PID that you want the message to be rescheduled on. The HTH header in the input message is then manipulating with this PID information before it is rescheduled. To do this, just enter the PID number in the above dialog box.

The Visual Dump Analyzer will handle the following three types of messages differently due to their unique nature:

- HTH Stand Alone Query messages
- HTH Session Query messages
- HTH Reply messages

HTH Stand Alone Queries will always execute on a PID from the Pseudo HTH PID pool assigned to the interface. Therefore, you have to select an unused PID from this pool and The Visual Dump Analyzer will create a dummy session on the PID by changing the HTH header from a Stand Alone Query to a HTH First in Series with a TPR matching the dummy session. This will force the HTH Query to execute on the PID selected by you.

The process for handling HTH Session Queries is almost the same. Here, The Visual Dump Analyzer will check to see if a session is already established for the TPR in question. If so, you are limited to rescheduling the input message on this PID number. If a session is not established, The Visual Dump Analyzer will create a new dummy session on the TPR from the HTH header – just like Stand Alone Queries. In this case, you are free to select any unused PID from the Pseudo HTH PID pool for the reschedule.



Because you will not know what PIDs are in the Pseudo PID pools associated with the interface, you will not be able to select a proper PID. Thus, you will have to select a PID at random or leave the original PID as the selected PID and try to reschedule the message. As a result, The Visual Dump Analyzer will likely reject the PID once the reschedule is attempted.

The rejection of the PID, however, will inform you of what PIDs are in the Pseudo PID pool. Therefore, once you have attempted one reschedule, you can change the PID number to a valid number from the pool and reschedule the input message a second time.

If The Visual Dump Analyzer is forced to establish a dummy session, it will add the letters "**VDA**" to the existing TPR field in HTH header. For example, if the original TPR was "**P00012345A**" it will be changed to "**PVDA0001235A**" to make it unique and easy identifiable. Also, be aware that The Visual Dump Analyzer will NOT remove any dummy TPRs that are produced. Instead, they will be left "active" and will be removed by Sys 11r2's normal clean-up process, which removes all TPRs that have been inactive for an extended period. The construction of dummy TPRs is another reason why The Visual Dump Analyzer does not allow reschedules to be performed on production systems.

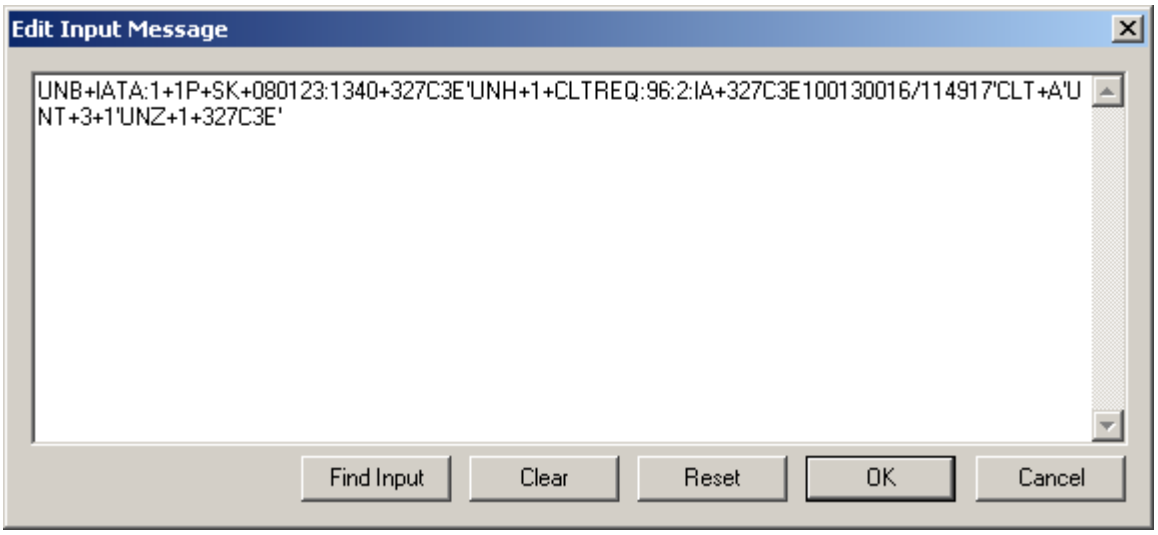
HTH reply message are simpler as they do not require being in a session. Instead, they will always execute on the PID number in the HTH header TPR field. Also, because The Visual Dump Analyzer can manipulate the TPR to force the rescheduled message to execute on any PID number, you can freely select its value.

Rescheduling Services

Because traces on services are set on the actual service and not on the PID it executes upon, the problem of determining the PID that is used for execution does not exist when rescheduling services. Be aware that when you have requested The Visual Dump Analyzer to set traces on the service as part of the reschedule logic, traces will be set for everyone using the service and not only for your execution. Thus, you should be careful to remove the traces shortly thereafter.

Editing or Finding the Input Message

Before rescheduling the input message, you can choose to edit it in order to gain additional information. This may be helpful, for example, if the error is due to a flawed input message. To edit the input message, press the "**Edit…**" button and the below dialog will be displayed.



In this dialog, you can either edit the input message or you can locate the input message in case it has not been located correctly. The Visual Dump Analyzer assumes that the input message is located in the Input MPA, however, that may not be the case for services. Thus, you can use "**Find Input**" to manually "load" the input message form any valid d-bank address within the dump.

Scheduling Edifact Messages directly using EDIT

When working with an Edifact message, you may not wish to have The Visual Dump Analyzer reschedule the input message as a true HTH message, thus avoiding the trouble of selecting a valid PID from the Pseudo PID pool. By checking the "**Schedule the message for the edifact handler**" box, you can request The Visual Dump Analyzer to remove the IATA HTH header and precede the input message with the "**EDIT**:" transaction code.

This will change the input message from a HTH message to a normal VDU transaction that is able to execute on any PID selected by you. Once you press "**Schedule Msg Now**", the transaction will be scheduled and the Edifact message will be routed directly to the Edifact Handler due to the "**EDIT**:" transaction code.

Setting TST Traces

When you press the "**Schedule Msg Now**" button, The Visual Dump Analyzer will automatically set the Breakpoint Traps or Snaps on the PID that you have set in the **Source**, **Variable** or **Data Windows**. You will then be informed of the status of the TST: settings as part of the response dialog.

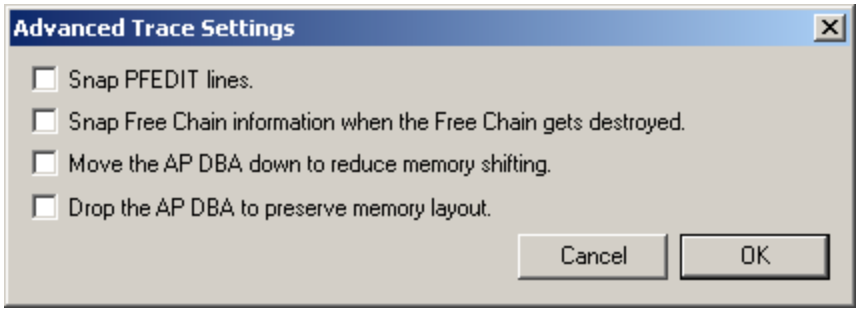
It is important that the version of the absolute element The Visual Dump Analyzer has found and used to calculate the actual Breakpoint Traps is the absolute that will be executed by the rescheduled transaction. This is because Traps and Snaps are set by using an absolute address and not a line number or variable name. Moreover, the correct absolute is required to convert the line number or variable name to the correct absolute address. The Visual Dump Analyzer will attempt to use the same version of the absolute element as the one used to analyze the dump. However, the correct absolute element for reschedules is always the version of the absolute that the rescheduled transaction will actually execute. This may or may not be the same version of the absolute found to be correct when analyzing the dump.

If the correct version of an absolute element is not found, the absolute address that is calculated to set Traps and Snaps will be flawed and then the Traps and Snaps will not be set in the correct locations of the executable code. This will cause very unpredictable results. Once the "**Reschedule Msg Now**" button has been pressed and the TST: settings have been placed, you can press "**Show TST flags**" to redisplay the result of the TST: settings and verify that the absolute element found was actually the correct one.

You can also set additional TST: flags manually on the service or PID from a normal HVTIP screen. The Visual Dump Analyzer will then add its TST: flags to those that are already set. For example, if you wish to reschedule an alternate library/bank, you can set the TST:ALTs on the PID or service as normal before rescheduling the input message. (See ***Setting Breakpoint Traps*** and ***Snapping data on Breakpoint Traps***)

Setting Visual Advanced Snaps

Before rescheduling a transaction, you can press the "**Advanced**" button to request The Visual Dump Analyzer to add **Visual Advanced Snaps** TST: trace flags. The TST: settings selected here will automatically be applied to the PID you have selected prior to rescheduling the input message.



In this dialog, you can also request PFEDIT output lines to be routed to the trace file. This provides a useful way of seeing where you are in the processing when reading the trace file. Note that for practical reasons, the width of each PFEDIT line is limited to 71 characters.

You can also request Visual Advanced Snaps to monitor the AWA Free Chain and report when it detects the chain to be destroyed. Once the AWA Free Chain is destroyed, **Visual Advanced Snaps** will place a line informing of this fact in the trace file. This will only be reported once in order to avoid filling the trace file with this identical warning statement.

Controlling the AP DBA

The AP DBA is used by Usas Sys 11r2 to hold the trace lines before they are written to the trace file. When traces are ON, Usas Sys 11r2 allocates the AP DBA as the first DBA in the AWA. It is important to realize that this causes the shifting of memory so that all other DBAs allocated in the AWA will be at different address locations than when traces are OFF. (See ***Memory Allocation and Tracing*** in ***User Guide for VISUAL TRACE And VISUAL PERFORMANCE ANALYSIS***).

Normally this is not a problem, but some complex problems tend to go away when traces are ON and re-appear when traces are OFF. Naturally, this makes finding this type of problem very difficult. This troubling behavior is often due to memory allocation and is therefore affected by the fact that the AP DBA is causing all other DBAs to be shifted down in the AWA. By setting "**Move the AP DBA down to reduce memory shifting**", **Visual Advanced Snaps** will force the AP DBA to be located at the very bottom of the AWA rather than at the top. The memory locations of all other DBAs are thereby preserved and hopefully the problem will then be traceable and reproducible when traces are ON. Note that the AP DBA is still present in the AWA when this option is set and will thus still affect the stack locations for HVTIP programs.

If forcing the AP DBA to the bottom of the AWA does not make the problem traceable, the only other solution is to completely drop the AP DBA. If you select "**Drop the AP DBA to preserve memory layout**", **Visual Advanced Snaps** will allow **Visual Breakpoint Traps** to be effective while preventing the AP DBA from being allocated. Further, you must ensure that any TST: trace flag that will place information in the trace file is turned OFF as this will also allocate the AP DBA. When the AP DBA is not allocated at all, the memory usage will be identical as to when traces are OFF and thus the problem will be reproducible.

Because the AP DBA is dropped, no trace file information will be captured by Usas Sys 11r2 resulting in an empty trace file. Instead, you will have to rely on **Visual Breakpoint Traps** to detect where and when the data gets destroyed. (See ***Forcing Aborts when a memory location gets destroyed***)

Removing TST Traces again

The Visual Dump Analyzer will automatically set all the needed TST: flags on a PID or service when the "**Reschedule Msg Now**" button is pressed, but it will not automatically remove them after the message is rescheduled. This is because removing the TST: flags would actually prevent you from capturing any trace data collected during the reschedule. So, to remove all TST: flags that are set by The Visual Dump Analyzer and you, press the "**Turn Traces OFF Now**" and a TST:DOFF transaction will be scheduled. It is important to remember to turn traces off once they are no longer needed – especially when traces are set on services or internal PIDs.

Setting Visual Breakpoint Traps

A trace is commonly used to determine the path taken by a transaction during its execution. Normal USAS tracing, however, only provides a trace that includes calls to ACB and FLSS routines. The volume of code that exists between ACB calls is not traced and therefore its path of execution is unknown. This problem is solved by **Visual Breakpoint Traps** (**TRAPS**)**.** Because **TRAPS** do not rely on any special events such as ACB calls for intercepting the execution, any piece of code can be analyzed in extensive detail.

Besides intercepting when a specific piece of code is executed, **Visual Breakpoint Traps** also allows you to gain control when a specific variable or address is referenced or updated. This is a very helpful way of finding code that alters or destroys specific data.

Among its many features, **TRAPS** can inform you whether you have executed a given line number or subroutine, it can snap data in various ways, it can force an abort at the "right" spot, and it can force an abort when a particular variable acquires a specified value. (See ***User Guide for VISUAL TRACE And VISUAL PERFORMANCE ANALYSIS***).

Technically, **TRAPS** are divided into two different types: I-Bank **TRAPS** and D-Bank **TRAPS**. I-Bank **TRAPS** are traps that occur whenever a specific I-Bank instruction is executed and D-Bank **TRAPS** are traps that that occur when a given D-Bank address is referenced or updated.

The Visual Dump Analyzer greatly simplifies the complex task of settings **Breakpoint Traps.** To set a trap, you only have to point at a specific location or piece of data where you want the trap to be set and right-click to open the Pop-up menu. Then, just select "**Set Breakpoint Trap**" and this will cause the location or data to be highlighted in **RED** indicating that a **Breakpoint Trap** has been set.

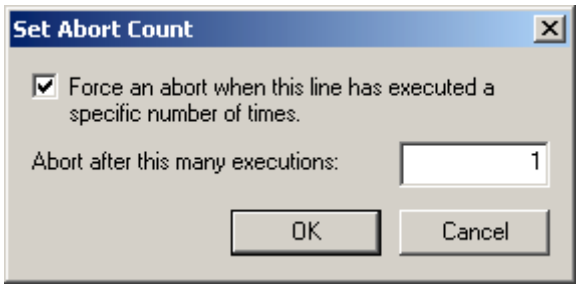
Once a **Breakpoint Trap** has been set, you can reschedule the input message again and The Visual Dump Analyzer will set the **TRAP** as requested before the reschedule. Of course, not all aborts can be reproduced by rescheduling the input message again, but a vast majority of aborts can be. With these, **Visual Breakpoint Traps** is a very effective way of finding the cause of an abort. (See ***Rescheduling the Input Message***)

Setting TRAPS on line numbers

In order to set a **TRAP** on a specific line number, just right-click the line you want to set the **TRAP** on. This will open a Pop-up menu where you can select "**Set Breakpoint Trap on line**". This will cause the **TRAP** to be set and the line will be highlighted in **RED** in the **Source Window** indicating that the **TRAP** is active.

```
0317.C*     THIS SUBROUTINE WILL SHOW THE CONTIGENCY ERROR TEXT    *
0318.C*     BY CALLING ERRPR$.                                     *
0319.C*                                                            *
0320.C*************************************************************
0321.C
0322.       SUBROUTINE SCONT
0323.C
0324.       DMPDDH = 1
0325.       LINE = DOTS
0326.       LINE(28:49) = ' CONTINGENCY ANALYSIS '
0327.       CALL PRGSAV('SAVE')
0328.C      ---------------------------------- EXPLAIN REQUESTE SPCERR CODE
0329.       IF (SPCERR.NE.0.OR.DMPTBL(CNTPKT).NE.0) THEN
0330.C        --------------------------------- DEBUG CODE FORCE ABORTS
0331.         IF (SPCERR.EQ.123456) THEN
0332.           SRCLIF = 1/SRCLIT              @ FORCE CONTINGENCY 5 (ZERO DIV)
0333.         ELSE IF (SPCERR.EQ.123457) THEN
0334.           LINE(4:SPCERR) = ' '           @ FORCE SUBSTRING ERROR
0335.         ELSE IF (SPCERR.EQ.123458) THEN
0336.           CALL XBCA(25,100000)           @ FORCE DBA ERROR
0337.CDML       IF (.FALSE.) THEN                  @ TEST CARGO DMS MACROS
0338.CBUG         CALL PRTADP('MYTEST',SPCERR,1)   @ TEST CARGO DEBUG MACROS
0339.CDML       ENDIF                              @ TEST CARGO DMS NACROS
0340.         ELSE IF (SPCERR.EQ.123459) THEN
```

By default, **TRAPS** that are set on line numbers will force a new abort the first time the line number is executed. However, you can alter this setting by right-clicking the **RED** line and selecting "**Set Abort Count…**". This will open the below dialog where you can control how many times the line must be executed before a new abort is forced or deselect the request for a new forced abort.



If the request to force a new abort is deselected, the line number **TRAP** will remain set and will result in the **TRAP** occurring every time the line is executed. In this case, the **TRAP** will not force a new abort, but it will instead snap any data you have requested to the trace file. Realize that the **TRAP** will remain set during the entire execution of the program, which may result in the **TRAP** occurring many times.

Setting a **TRAP** on a line number in an ACB or FLSS routine is done in the same way as normal HVTIP programs.

When a line number **TRAP** is set, it is normally placed on the first assembler instruction generated by the FTN line number. If the FTN line does not actually cause any assembler instructions to be generated (e.g., Comment lines, ENDIF and CONTINUE statements), the **TRAP** will be rejected indicating that you should set the **TRAP** on the next FTN line that will generate executable instructions.

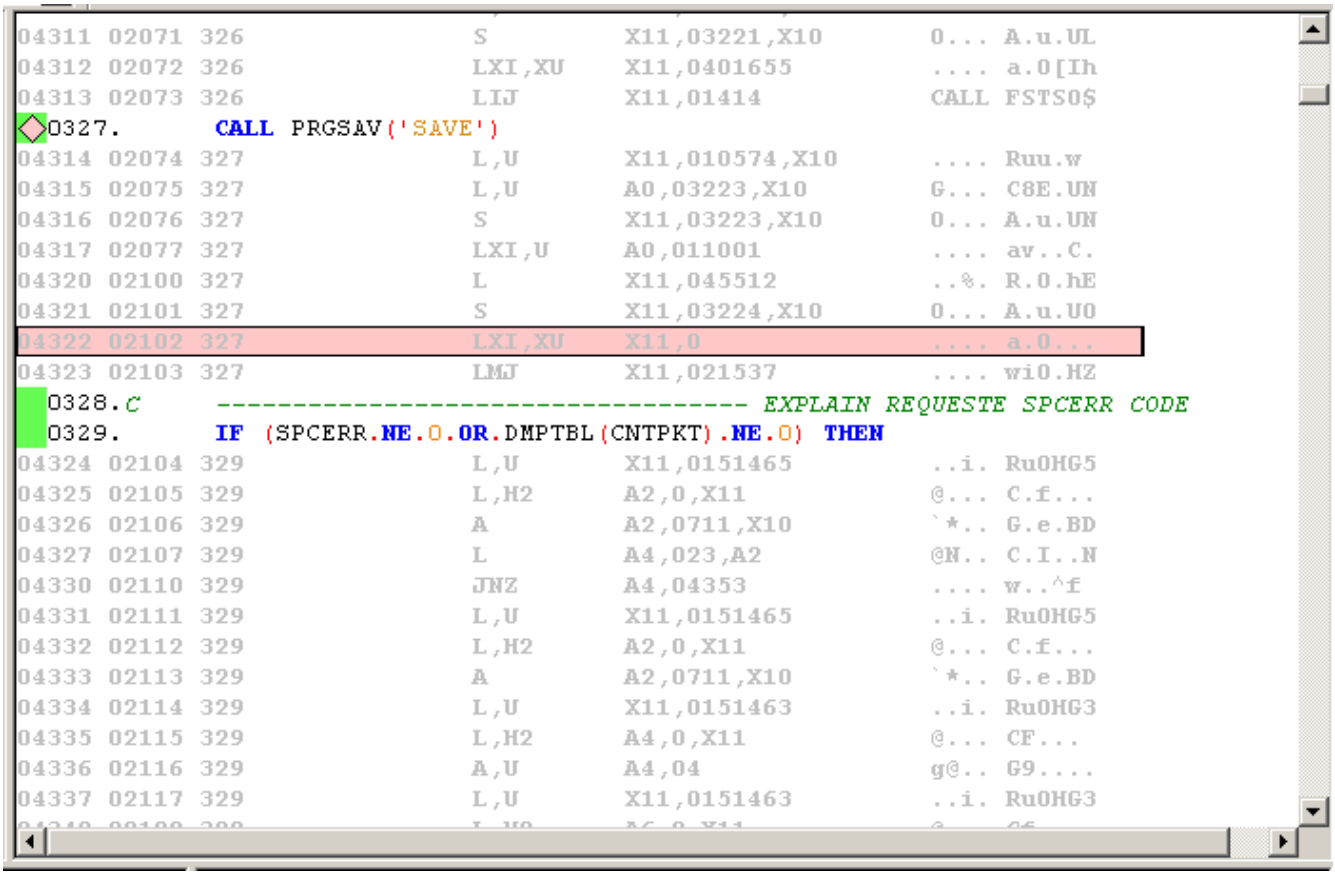
If the Z-option (optimization) is used with the FTN compiler, assembler code from multiple FTN lines can be merged in order to generate more efficient code. This can cause a line number **TRAP** to be set some instructions away from where it would appear to be set if compared to the source code. Usually this is not a problem, but if you want to be in complete control of the exact location of your **TRAP**, use "**Show MASM code**" to see the assembler code and set the **TRAP** on the specific MASM instruction you want rather than on a line number. (See ***Setting TRAPS on MASM instructions***)

When setting **TRAPS** on an IF-statement, it can sometimes be difficult to predict if the instruction is actually being executing and therefore should be avoided.

To clear the **TRAP,** again right-click on the **RED** line and select "**Clear Breakpoint Trap**" and the **TRAP** will be cleared. Alternatively, on the main menu you can select "**Tools | Clear Breakpoint Trap**".

Setting TRAPS on MASM instructions

Setting **TRAPS** on an FTN line number will attempt to set the **TRAP** on the first instruction generated by the specified line number. When the FTN Z-option is used, however, the compiler sometimes merges instructions from multiple FTN lines in order to optimize the execution and thus the **TRAP** may not be set on the exact instruction that is wanted. To set a **TRAP** on a specific instruction, right-click anywhere in the source window and select "**Show MASM code**". Once the masm code is shown, simply right-click the desired masm instruction and select "**Set Breakpoint Trap on line**". This will cause the **TRAP** to be set on this precise absolute address.



As with line number **TRAPS**, it is default is to force a new abort the first time an instruction is executed. This can also be altered in the same way by right-clicking the **RED** line and selecting "**Set Abort Count…**".

Setting TRAPS on internal subroutines

As an alternative to setting a **TRAP** on a specific line number, The Visual Dump Analyzer allows you to set a **TRAP** on an internal FTN subroutine. A **TRAP** that is set on an internal subroutine will cause the **TRAP** to occur on both the entry to and the exit from the subroutine. Setting a **TRAP** on an internal subroutine differs greatly from manually setting the **TRAP** on the line number of the subroutine statement. This is because of the way FTN enters and exits subroutines. A **TRAP** that is manually set on the line with the subroutine statement will cause the **TRAP** to only occur on the exit from the subroutine and not on the entry to the subroutine.

Requesting a **TRAP** to occur on both the entry and exit from an internal subroutine combined with the snapping of data is a very effective way of verifying what changes an internal subroutine makes to the snapped data. The trace will show the values of the snapped data on both entry to and exit from the subroutine regardless of the RETURN statement that is used to exit. This is even true if the subroutine contains multiple RETURN statements or if the subroutine uses an error return (i.e., RETURN n) to exit.

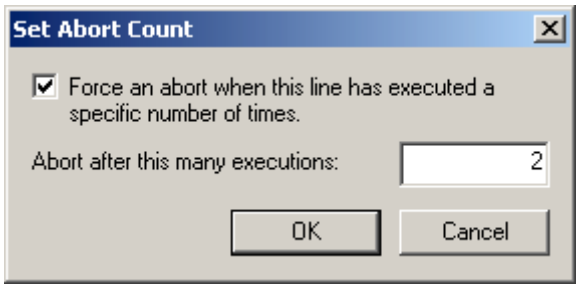
Realize that the main routine in an HVTIP program or mapped-in relocatable are technically also internal subroutines and thus you can also set a **TRAP** on the entry and exit from these routines. That is, you can set a **TRAP** that will occur every time your program is called and every time your program makes a return.

To set a **TRAP** on a subroutine, right-click anywhere in the **Source Window** showing the subroutine and select "**Trap calls/returns to <*Subroutine name*>**". This will cause the **TRAP** to be set on all calls to and returns from the subroutine while highlighting the SUBROUTINE statement in RED indicating the **TRAP** is active.

```
0319.C*                                                              *
0320.C*************************************************************
0321.C
0322.      SUBROUTINE SCONT
0323.C
0324.      DMPDDH = 1
0325.      LINE = DOTS
0326.      LINE(28:49) = ' CONTINGENCY ANALYSIS '
0327.      CALL PRGSAV('SAVE')
0328.C     ---------------------------------- EXPLAIN REQUESTE SPCERR CODE
0329.      IF (SPCERR.NE.0.OR.DMPTBL(CNTPKT).NE.0) THEN
0330.C       --------------------------------- DEBUG CODE FORCE ABORTS
0331.        IF (SPCERR.EQ.123456) THEN
0332.          SRCLIF = 1/SRCLIT              @ FORCE CONTINGENCY 5 (ZERO DIV)
0333.        ELSE IF (SPCERR.EQ.123457) THEN
0334.          LINE(4:SPCERR) = ' '           @ FORCE SUBSTRING ERROR
0335.        ELSE IF (SPCERR.EQ.123458) THEN
0336.          CALL XBCA(25,100000)           @ FORCE DBA ERROR
0337.CDML      IF (.FALSE.) THEN                  @ TEST CARGO DMS MACROS
0338.CBUG        CALL PRTADP('MYTEST',SPCERR,1)   @ TEST CARGO DEBUG MACROS
0339.CDML      ENDIF                              @ TEST CARGO DMS NACROS
0340.        ELSE IF (SPCERR.EQ.123459) THEN
0341.          IDX = H1(CBAREA(2))-ADDR$(CBAREA(1))-1  @ 1ST FREE ENTRY
0342.          CBAREA(IDX) = 0                @ DESTROY FREE CHAIN
0343.          CBAREA(IDX+1) = 0              @ DESTROY FREE CHAIN
0344.          CBAREA(IDX+2) = 0              @ DESTROY FREE CHAIN
0345.          CALL XBCA(25,100)              @ CAUSE THE ABORT
```

It is default for internal subroutine **TRAPS** to force an abort on the first exit from the routine. In the same way as line number or MASM instruction **TRAPS**, you can alter this setting by right-clicking the RED line and selecting "**Set Abort Count…**".

Note that the default setting for the abort count is 2 rather than 1. This is because FTN technically uses the SUBRUOTINE statement to generate both the entry and return code.



Because of this, The Visual Dump Analyzer controls the execution (hit) count by tallying the number of times either the entry code or the return code from the SUBROUTINE statement is executed. This results in every CALL to the subroutine to make 2 hits – 1 on entry and 1 on exit. As a natural result of this, all the odd hits will be entry hits whereas all the even hits will be exit hits. Moreover, the default value of 2 will force an abort on the first return from the subroutine. Finally, by using this method, you can calculate the hit count that is needed to force an abort on the precise entry or exit of the subroutine that you want.

As with line number **TRAPS**, you can also use this dialog to turn the forcing of new aborts completely OFF by deselecting the "**Force an abort when…**". This will still leave the **TRAP** active and cause any snaps to be written to the trace file on every entry and exit from the routine.

Setting TRAPS on HVTIP calls

When an abort occurs in a program, it is often not because the error occurs where the abort is taken, but instead because it is at this specific point in the program logic that an error is recognized and continued execution is either pointless or impossible.

In these cases, usually the initial task is to identify the program that caused the error, which may or may not be the same program that triggered the abort. To assist you in finding the program that actually caused the error, The Visual Dump Analyzer allows you to set a **TRAP** that will occur when your program calls another HVTIP program and when the called HVTIP program returns back to yours.

```
0403.         ELSE
0404.           SRCLIF = H1(DMPTBL(CNTPKT))
0405.           BITS(SRCLIF,19,1) = 0          @ CLEAR EM ASYNC INDICATOR
0406.         ENDIF
0407.         CALL SEDOUT
0408.C        ---------------------------------- ALLOW USER TO SET X11
0409.         IF (USRX11.EQ.1) CALL SUX11
0410.C        ---------------------------------- GET CGY TEXT
0411.         FILE = HELPFL
0412.         PRG = 'SED$CTG'
0413.         FDPRG = FD(PRG(1:6))
0414.         FDPRG2= FD(PRG(7:12))
0415.         FDVER,FDVER2 = FDSPAC
0416.         CALL SEDSRC(FILE,LINE)
0417.         IF (SRCCNT.EQ.0) THEN
0418.           IF (SPCERR.NE.0) THEN
0419.             CALL POCTAL(SPCERR,WORKLN)
0420.           ELSE
0421.             CALL POCTAL(DMPTBL(CNTPKT),WORKLN)
0422.           ENDIF
0423.           LINE = 'UNKNOWN CONTINGENCY CODE ' & WORKLN(1:12)
0424.           SRVSEG = BOOL('CTG ')
0425.           SRVTXT = 1
0426.           CALL SEDOUT
0427.         ENDIF
0428.C        ----------------------------------------------
```

In this example, the **TRAP** will occur both when SEDOUT is called and returns in line 407, also when SEDSRC is called and returns in line 416, and finally when SEDOUT is called and returns in line 426. Moreover, these call and return **TRAPS** will occur with *every* HVTIP program that is called by your program. By combining these **TRAPS** with the snapping of data, you can gain valuable information in determining the program that causes the error situation resulting in an abort.

To set an HVTIP **TRAP,** right-click in the **Source Window** to open the Pop-up menu and select "**Trap calls/returns to HVTIP**".

The default abort count setting for HVTIP **TRAPS** is 1, which will force an abort on the first call to another HVTIP program. As with subroutine **TRAPS**, each CALL will result in 2 hits – one when the HVTIP program is called and a second when it returns. Again, this means that all odd hit counts are calls and all even counts are returns. Realizing this, you can calculate the HVTIP hit count needed to force an abort at the desired time. (See ***Setting TRAPS on internal subroutines***)

Setting Visual Breakpoint Traps on data

The Visual Dump Analyzer allows you to set **Visual Breakpoint TRAPS** on any data location using the symbolic name of the location whenever possible. This can quickly provide you information on where a variable is used or updated within a transaction's execution.

Technically, all **TRAPS** have to be set using a fixed absolute address, however, The Visual Dump Analyzer uses its knowledge of USAS to calculate, set and even relocate the **TRAPS** when necessary. For example, if you set a **TRAP** on a DBA, the **TRAP** will naturally become active when the DBA is allocated. If, however, the DBA is later expanded causing it to move location, The Visual Dump Analyzer will ensure that the actual **TRAP** address is automatically recalculated and reset to match the new location of the DBA. The same is true for variable **TRAPS.** If a **TRAP** is set on a variable in a program that is called repeatedly, the variable's stack may not be allocated at the same location each time the program is called. Thus, The Visual Dump Analyzer will recalculate the absolute **TRAP** address and set it to align with the active stack address.

Continuing, The Visual Dump Analyzer will simulate an update to the variable when it is initially "created" so its initial value is displayed when it first exists on the stack. This is done because a variable's type and form of initiation (if any) may not cause a true update Breakpoint occurrence. By simulating a Breakpoint update hit, however, you will always be able to see what the initial value of a variable is when a program starts execution.

A simulated breakpoint update hit will also be simulated for any I/O read operation that "updates" the **TRAP** address in a DBA. That is, I/O reads to DBAs that cause the **TRAP** address to "change" are handled as normal **TRAP** hits even though the Exec will actually not cause a true Breakpoint interrupt for this type of update. To USAS programmers, however, these updates are just as relevant as any other updates to the DBA.

Because the Exec requests a **TRAP** to be a single full-word absolute address, The Visual Dump Analyzer will always convert the symbolic **TRAP** to be a single full-word absolute address even if the **TRAP** is only for a partial word. For example, if the variable to **TRAP** is only one character stored within Q1 of a word, the **TRAP** will actually be set for the entire word. Therefore, any updates to Q2, Q3 and Q4 of this word will also cause the **TRAP** to hit even though the variable was actually not updated. For FTN program with STD66 this is not a problem, but for character tables using STD77, this needs to be taken into account.

Similarly, if a **TRAP** is set on a character variable larger than 4 characters – thus taking up more than one word on the stack – only the first word will be under the control of the **TRAP** with updates to the 2nd and later words not causing the **TRAP** to hit.

These limitations are due to the Breakpoint logic supported by the Exec and therefore cannot be circumvented by The Visual Dump Analyzer.

For all **TRAPS** to data areas, you must decide if the **TRAP** should hit only on UPDATE references or on ALL references (i.e., read and update). Read references are defined to be any source code statement that references a variable on the right side of the equals sign ("=") within an assign statement or uses the variable within a test IF-statement. That is, read references include all references that the source code makes to a variable with actually storing a value into it.

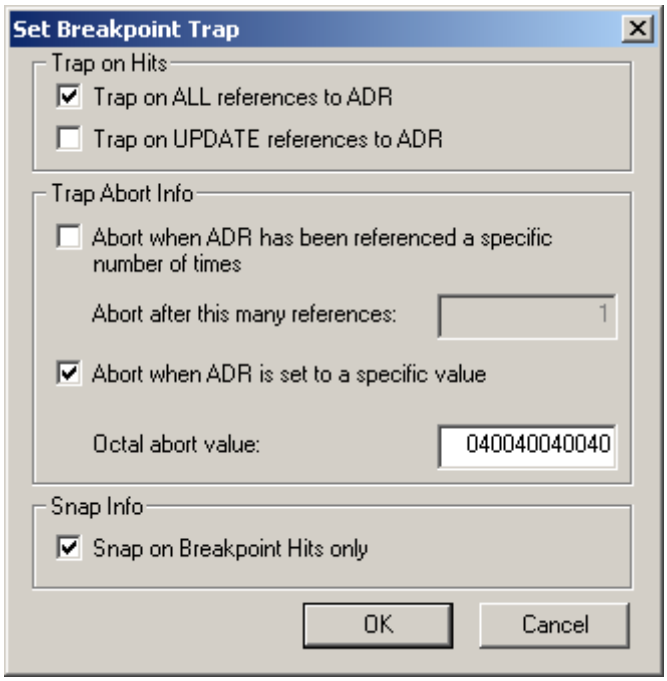
Continuing, update references are those references that actually store a value into the variable. Even when a variable has the same value before the store as it did after, it will still cause an update **TRAP** hit. For example, update references refer to variables that appear on the left side of the equals sign ("=") within an assign statement and to variables used within a CALL statement's arguments when the called routine stores/returns a value into the variable.

Realize that update references refer to all updates to a variable, even if the store operation was not actually intended. That is, when a flaw exists within the logic resulting in an unintended update to a variable that is the target of a **TRAP**, the **TRAP** contingency will still be triggered. This could occur, for example, when a 100-word table is defined and the program accidentally stores a value into the table using an index less than 1 or greater than 100. In this case, FTN will produce a flawed address calculation and store the value outside of the address range occupied by the table. As a second example, an unintended update could occur when a variable within a DBA is updated, but the DBA is not actually allocated.

These unintended updates are normally very difficult to remedy, but The Visual Dump Analyzer makes them much easier to find by allowing you to set **TRAPS** on the variable that gets destroyed.

Setting TRAPS on variables

To set a **TRAP** on a variable, right-click the variable name in the **Variable Window** to open the Pop-up menu and select **"Set Breakpoint Trap…**". In this example, the variable named "ADR" was clicked on. As a result, the below dialog box is opened allowing you to define the **TRAP** settings.



Once this dialog is displayed, you must first decide whether you want the TRAP to hit on ALL references (i.e., read and update references) or only on UPDATE references. Next, you need to consider whether The Visual Dump Analyzer should or should not abort the transaction at a specific TRAP hit. If you choose to not abort the transaction at a given TRAP hit, all the hits will be written to the trace file resulting in an informative trace of the scheduled transaction.

If you choose for the transaction to be aborted, you must select one of two options in the "**Trap Abort Info**" section. This includes either requesting the transaction to be aborted when, first, the **TRAP** variable has been referenced a specified number of times or, second, when the **TRAP** variable obtains a specific value.

Determining the specific **TRAP** hit that you would like an abort to occur can sometimes be difficult. To assist with this, sometimes it is helpful to reschedule the transaction with traces on and without an abort request. Then, in the trace file you will find all the **TRAP** hits that have occurred along with the hit number for each of them. By scanning the trace file, you can easily identify the **TRAP** hit count that results in erroneous data to appear. Next, after setting the hit count within the above dialog, you can reschedule the transaction thus triggering an abort at this point of interest.

Finally, you need to consider how often you want the snap information to be written to the trace file. By default, a **TRAP** will only cause the **TRAP** variable to be snapped (written) to the trace file on **TRAP** hits, thus providing you a history of references to the variable. In addition to **TRAP** hits, you can also request each CALL to and RETURN from an ACB or FLSS routine to cause a snap of the variable. To request The Visual Dump Analyzer to snap the variable on all ACB and FLSS routine activity, deselect the "**Snap on Breakpoint Hits only**" in the "**Snap Info**" area.

At times, this TRAP feature can produce very large trace files, but this volume of data can often be invaluable in resolving some of the most difficult errors and aborts.

Once the **TRAP** information is completely defined, you must press the "**OK**" button and the variable **TRAP** will be set. To inform you of the active **TRAP**, the variable will be highlighted in **RED** indicating the **TRAP** is active.

Variable	Type	Scope	Size	Address	Format	Value
IDX	INTEGER	LOCAL	4	516133	Ascii	*PCI
SZ	INTEGER	LOCAL	4	516120	Ascii	USAS
ACBBDI(24,3)	INTEGER	GLOBAL	4	516216	Octal	0401655401655
ADR	INTEGER	GLOBAL	4	515773	Octal	040040040040
CALABS	CHARACTER*12	GLOBAL	12	515543	Ascii	
CALREL	CHARACTER*12	GLOBAL	12	515745	Ascii	
CBA	INTEGER	GLOBAL	4	515775	Decimal	0
CQ	INTEGER	GLOBAL	4	515764	Decimal	0
DOTS	CHARACTER*80	GLOBAL	80	516067	Ascii	....
EDITFC	CHARACTER*4	GLOBAL	4	516064	Ascii	EDI:
FDOS	INTEGER	GLOBAL	4	515755	Decimal	21878821189
FDSPAC	INTEGER	GLOBAL	4	516122	Decimal	5453926725
FILE	CHARACTER*40	GLOBAL	40	516052	Ascii	
FTNLIN	CHARACTER*8	GLOBAL	8	515634	Ascii	
GETHNT	LOGICAL	GLOBAL	4	515661	Logical	FALSE
HELPFL	CHARACTER*40	GLOBAL	40	515733	Ascii	USAS *SED $FILE

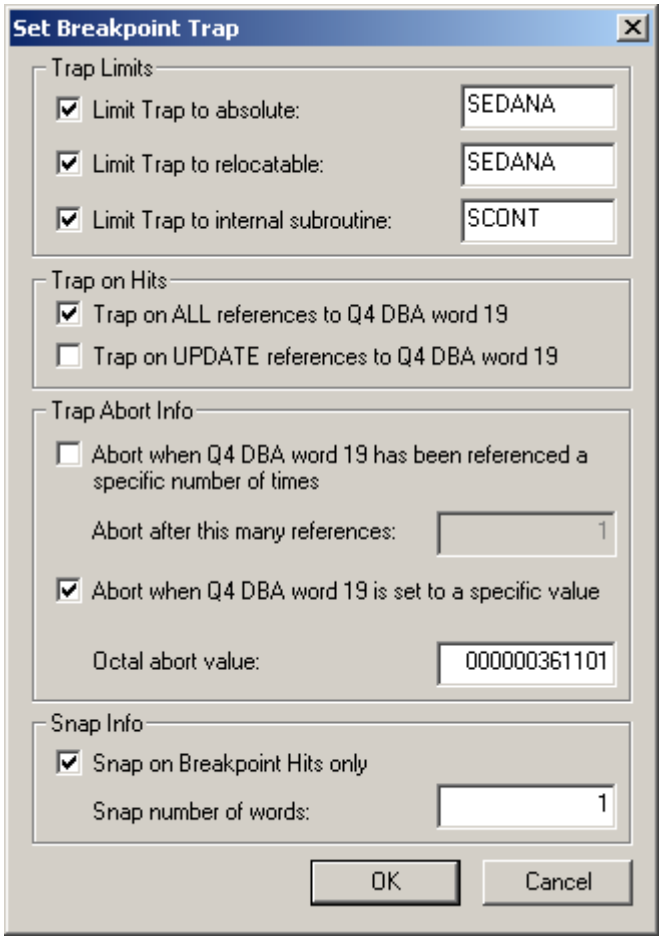
Even though a **TRAP** will only be set on one word, you can request more words to be snapped to the trace file. By default, the number of words snapped is set to the variable's word size, but you are allowed to update this value. To change the number of snapped words, right-click the **RED** variable line and select "**Set number of Words to Snap…**". This will open a dialog allowing you to enter the number of words to be written to the trace file for each snap. For example, if you set a **TRAP** on a string variable called LINE that is 80 characters long, the **TRAP** will only be set for the first word, but you can request any or all of the 20 words to be snapped. Similarly, you can set a **TRAP** on the first word in a table while requesting the snap to show any portion or all of the table.

To remove a **TRAP,** simply right-click the **RED** variable line, select "**Clear Breakpoint Trap/Snap**" and the **TRAP** will be removed. Alternatively you may select "**Tools | Clear Breakpoint Trap**" in the main menu to clear it. Because only one **TRAP** can be set at a time, setting a new **TRAP** will automatically cause an old **TRAP** to be cleared.

Setting TRAPS on DBAs or PDBs

Setting a **TRAP** for a DBA or a PDB works similar to setting a **TRAP** for a variable, except that you cannot set a **TRAP** using the symbolic names of defines. This is because the FTN compiler does not register defines to have specific addresses that are predetermined before execution. Instead, the FTN compiler creates the specific MASM instructions needed to locate the defines each time they are used. This prevents The Visual Dump Analyzer from calculating the exact address that relates to a given define beforehand. Thus, **TRAPS** for DBAs or PDBs cannot be set by name (symbolically); rather they have to be set on a specific "word" within the PDB or DBA.

To set a **TRAP** for a DBA or a PDB, you must right-click the actual word in the DBA or PDB that you want the **TRAP** to be set on. This will open a Pop-up menu allowing you to select "**Set Breakpoint Trap…**", which will open the below dialog where you can set and customize the **TRAP**.



Setting a **TRAP** on a variable naturally causes the **TRAP** to be active only when the variable is in scope. When setting **TRAPS** for DBAs or PDBs, the scope is from when the DBA or PDB is first allocated until it is released by the XBCF routine. Unlike the scope of local variables that only span one program, DBA or PDB **TRAPS** will often span many programs and will thus cause **TRAP** hits outside the program you are interested in. To restrict the duration that a **TRAP** is active, you can use the "**Trap Limits**" area. Here you can define the boundaries of a trap by specifying an absolute program, specifying a relocatable, or even specifying an internal FTN subroutine. If you deselect these scope limits, the **TRAP** will be active whenever the DBA or PDB is allocated.

The "**Trap on Hits**" and "**Trap Abort Info**" areas are identical to those for variable **TRAPS**. The "**Snap Info**" area, however, has an extra field allowing you to set the snap size directly. This is because, even though the **TRAP** will only be set for the specified word, you can request more words to snapped and written to the trace file with each **TRAP** hit. A quick way to set the number of words to be snapped is to select the words (using the normal Windows selection method) that you want to be snapped in the **Data Display Window** before right-clicking the first word. This will inform The Visual Dump Analyzer of the number of words you want snapped and preset this in this dialog when it is opened.

Once you press "**OK**", the **TRAP** is set and the location of the TRAP will be highlighted in **RED** indicating that it is active.

To remove the **TRAP**, select "**Tools | Clear Breakpoint Trap**" in the main menu or right-click the **RED** word and select "**Clear Breakpoint Trap/Snap**". Because only one **TRAP** can be set at a time, setting a new **TRAP** will automatically cause an old **TRAP** to be cleared.

Setting TRAPS on an fixed address

TRAPS that are set on a fixed address are defined and execute exactly as **TRAPS** on DBAs and PDBs. To do this, drag the common block containing the fixed address to a **Data Display Window** and right-click the address you want the **TRAP** to be set on. Then, the same dialog as for DBA and PDB **TRAPS** will be opened allowing you to set and customize the **TRAP**.

Forcing Aborts when a memory location gets destroyed

Some of the most complicated aborts to solve are when a flawed process corrupts data that doesn't belong to it. This type of memory destruction is normally very difficult to find as it is likely to cause aborts very far from where the data is actually destroyed – often in a completely different process. Memory that is destroyed in this manner can relate to common blocks, variables or defines in DBAs.

Visual Breakpoint Traps makes this type of problem much easier to solve. To do this, first set a Variable, DBA or fixed address **TRAP** for the memory location that is being destroyed and then reschedule the transaction. Even though you have not requested the **TRAP** to abort the transaction, the trace file will show a hit every time the memory location was updated. Often times this is enough to solve the problem as the trace will show exactly where the memory location was updated, what it was updated to, and the code or process that performed the update.

Some very complex problems, however, can be affected by the fact that traces are turned ON and thus the data location that you are monitoring no longer becomes corrupted. Instead, the corrupt data may now be at a completely different memory location. As a consequence, the trace will not show anything wrong, but will only inform you of valid updates to the snapped memory location. (See ***Controlling the AP DBA*** *and* ***Memory Allocation and Tracing*** in ***User Guide for VISUAL TRACE*** and ***VISUAL PERFORMANCE ANALYSIS***)

Visual Breakpoint Traps will aid you with this situation as it enables you to run a trace without producing a trace file, thereby not altering the memory layout for the transaction. Since you do not get a trace file to look at, you will have to force a new abort when the "bad" value is stored in the memory location.

Before running the transaction, you must inform **Visual Breakpoint Traps** to not produce a trace file. You do this by pressing the "**Advanced**" button in the reschedule dialog and selecting "**Drop the AP DBA to preserve memory layout**". (See ***Setting Advanced TST Traces***)

Now, reschedule the transaction while ensuring that you have turned OFF all other TST flags that will cause data to be written to the trace file as it will cause the trace DBA (AP) to be allocated and thereby change the memory layout. This time you should get an abort informing you of the specific code that updated the memory location that contains the "bad" value.

Setting Visual Breakpoint Snaps

The Visual Dump Analyzer allows you to set a **SNAP** on any memory location causing its value to be written to the USAS trace file as a separate "**VDA***" **SNAP** line. You can set **SNAPS** on Variables, DBAs, PDBs, Basic Mode Addresses and Extended Mode Virtual Addresses.

The Visual Dump Analyzer will write the **SNAP** lines to the trace file when one of four unique events occur:

1) BEFORE SNAPS - each time an ACB or FLSS routine is called.
2) AFTER SNAPS - each time the called ACB or FLSS routine returns to the main HVTIP program.
3) AFTER BRKPT - each time a **Visual Breakpoint TRAP** hits.
4) STACK ALLOCATION – when a variable is initially generated on the program stack.

Each of these situations will cause The Visual Dump Analyzer to write a **SNAP** line to the trace file. The **SNAP** line in the trace file varies slightly based upon the type of data that is being written, but they all display the data value of the **SNAP** along with the event that caused it to occur.

Consider the following source line:

```
2225      CALL PFMA07($80001,INPUTI(1),CHARS,NEXT)
```

If a **SNAP** is defined on the NEXT variable, it will cause two **SNAP** lines to be written to the trace file. First, a BEFORE PFMA07 line will inform of its value before the PFMA07 call. Second, an AFTER PFDM07 line will inform of its value when PFMA07 has returned. Below you will note that the NEXT variable was changed to the value of 003 by the PFMA07 routine.

```
0024.VDA*VAR  NEXT = 000000000000  ADR:140300  BEFORE PFMA07
0025.****CALL SYSACB:  PFMA07  CALLED FROM PX7AGS LINE 2225
0026.VDA*VAR  NEXT = 000000000003  ADR:140300  AFTER  PFMA07
```

By displaying **SNAPS** before and after an ACB or FLSS routine is executed, you can easily note whether a particular piece of data was changed by it.

In the below example, the **Visual Breakpoint TRAP** has been set on the execution of line 377. This will cause the **SNAP** lines in the trace file to be displayed as follows:

```
0002.VDA*VAR  NEXT = 000000000000  ADR:140300  AFTER STACK ALLOCATION
0003.VDA*BRKP BREAK POINT HIT NO 1      IN PX7AGS LINE 377
0004.VDA*VAR  NEXT = 000000000000  ADR:140300  AFTER BREAK POINT
```

Note that because the **SNAP** data is a variable (NEXT), The Visual Dump Analyzer has added an additional line displaying the value of NEXT right after the stack allocation to inform you of its initial value.

Be aware that ACB and FLSS calls can be nested so that one ACB routine may call other ACB routines. When this happens, you will see both the BEFORE and AFTER **SNAP** on the initial ACB routine call, but for technical reasons only BEFORE **SNAPS** will be produced for the nested ACB calls. However, **SNAPS** for XFSxx and XDFxx I/O routines will always be shown with both BEFORE and AFTER lines even when they are within nested calls.

Setting SNAPS on variables

To set a **SNAP** for a variable, simply right-click the variable name in the **Variable Window** to open the Pop-up menu and select "**Set Variable Snap**".

Variable	Type	Scope	Size	Address	Format	Value
IDX	INTEGER	LOCAL	4	516133	Decimal	5658150473
SZ	INTEGER	LOCAL	4	516120	Decimal	11430298195
ACBBDI(24,3)	INTEGER	GLOBAL	4	516216	Decimal	-34112928850
ADR	INTEGER	GLOBAL	4	515773	Decimal	4303372320
CALABS	CHARACTER*12	GLOBAL	12	515543	Ascii	
CALREL	CHARACTER*12	GLOBAL	12	515745	Ascii	
CBA	INTEGER	GLOBAL	4	515775	Decimal	0
CQ	INTEGER	GLOBAL	4	515764	Decimal	0
DOTS	CHARACTER*80	GLOBAL	80	516067	Ascii	....
EDITFC	CHARACTER*4	GLOBAL	4	516064	Ascii	EDI:
FDOS	INTEGER	GLOBAL	4	515755	Decimal	21878821189
FDSPAC	INTEGER	GLOBAL	4	516122	Decimal	5453926725
FILE	CHARACTER*40	GLOBAL	40	516052	Ascii	
FTNLIN	CHARACTER*8	GLOBAL	8	515634	Ascii	
GETHNT	LOGICAL	GLOBAL	4	515661	Logical	FALSE
HELPFL	CHARACTER*40	GLOBAL	40	515733	Ascii	USAS *SFD $FILE

This will cause the variable to be highlighted in **GREEN** indicating that the **SNAP** is active. By default, the **SNAP** will contain the number of words occupied by the variable, however, you can change this by right-clicking the **GREEN** line and selecting "**Set Number of Words to Snap…**".

The **SNAP** can be removed by again right-clicking the **GREEN** line and selecting "**Clear Breakpoint Trap/Snap**".

Setting SNAPS on DBAs, PDBs or a fixed address

To set a **SNAP** for a DBA, PDB or a fixed memory address, drag the data to the **Data Display Window** and right-click the word you want to **SNAP.** This will open a Pop-up menu where you can select "**Set Snap**" and the **SNAP** will be set.

```
08 DBA
0000 733600000003 114113102101 102123056040 040040040040 ....LKBABS.
0004 040040040040 040040040040 040040040040 040040040040
0010 040040040040 040040040040 040040040040 040040040040
0014 040040040040 040040040040 040040040040 040040040040
0020 040040040040 040040040040 040040040040 040040040040
0024 040040040040 040040040040 040040040040 040040040040
0030 040040040040 040040040040 040040040040 040040040040
0034 040040040040 040040040040 040040040040 040040040040
0040 040040040040 040040040040 040040040040 040040040040
0044 040040040040 040040040040 040040040040 040040040040
0050 040040040040 040040040040 040040040040 040040040040
0054 040040040040 040040040040 040040040040 040040040040
```

Due to the nature of defines, you cannot set a **SNAP** directly on a DBA or PDB define by using its symbolic name. Instead, you must calculate its address location and then set the **SNAP** on the relative word in the DBA.

You can change the number of words to **SNAP** by right-clicking the **GREEN** words and selecting "**Set Number of Words to Snap…**". Alternatively, you can use the normal Windows method by selecting all the words you wish to **SNAP**, then right-clicking on the chosen words to open the Pop-up menu, and finally selecting "**Set Snap**".

Dynamic PRTADP calls

To set a Dynamic PRTADP call, just set a **TRAP** on a line number and enter the **SNAP** data that you would like to be displayed when the **TRAP** occurs. This method is simple and effective and allows you to dynamically produce **SNAPS** that work just like PRTADP calls, but without having to update and recompile your program.

```
0328.C      ---------------------------------- EXPLAIN REQUESTE SPCERR CODE
0329.      IF (SPCERR.NE.0.OR.DMPTBL(CNTPKT).NE.0) THEN
0330.C       -------------------------------- DEBUG CODE FORCE ABORTS
0331.        IF (SPCERR.EQ.123456) THEN
0332.          SRCLIF = 1/SRCLIT                @ FORCE CONTINGENCY 5 (ZERO DIV)
0333.        ELSE IF (SPCERR.EQ.123457) THEN
0334.          LINE(4:SPCERR) = ' '             @ FORCE SUBSTRING ERROR
0335.        ELSE IF (SPCERR.EQ.123458) THEN
0336.          CALL XBCA(25,100000)             @ FORCE DBA ERROR
0337.CDML      IF (.FALSE.) THEN                    @ TEST CARGO DMS MACROS
0338.CBUG        CALL PRTADP('MYTEST',SPCERR,1)     @ TEST CARGO DEBUG MACROS
0339.CDML      ENDIF                                @ TEST CARGO DMS MACROS
0340.        ELSE IF (SPCERR.EQ.123459) THEN
0341.          IDX = H1(CBAREA(2))-ADDR$(CBAREA(1))-1   @ 1ST FREE ENTRY
0342.          CBAREA(IDX) = 0                  @ DESTROY FREE CHAIN
0343.          CBAREA(IDX+1) = 0                @ DESTROY FREE CHAIN
0344.          CBAREA(IDX+2) = 0                @ DESTROY FREE CHAIN
0345.          CALL XBCA(25,100)                @ CAUSE THE ABORT
```

Variable	Type	Scope	Size	Address	Format	Value
IDX	INTEGER	LOCAL	4	516133	Decimal	5658150473
SZ	INTEGER	LOCAL	4	516120	Decimal	11430298195
ACBBDI(24,3)	INTEGER	GLOBAL	4	516216	Decimal	-34112928850
ADR	INTEGER	GLOBAL	4	515773	Decimal	4303372320
CALABS	CHARACTER*12	GLOBAL	12	515543	Ascii	
CALREL	CHARACTER*12	GLOBAL	12	515745	Ascii	
CBA	INTEGER	GLOBAL	4	515775	Decimal	0
CQ	INTEGER	GLOBAL	4	515764	Decimal	0
DOTS	CHARACTER*80	GLOBAL	80	516067	Ascii	....

In this example, the **TRAP** has been set on line 334 and the ADR variable has been requested to be **SNAPPED** when line 334 executes. Now, all you have to do is reschedule the input message with this setting and examine the trace file. The trace file will now contain a Breakpoint Hit for line 334 along with a **SNAP** line giving the value of ADR every time it was executed. (See ***Rescheduling the Input Message***)

View Construction

A view is an element in a program file that contains freeform text and expressions related to the dump and absolute. When a *view* is displayed, the expressions are resolved and their values are replaced within the freeform text of the *view*. In order for the *view* process to recognize an expression, it is always enclosed within square brackets (i.e., *[expression]*).

Following is an example of a simple *view*:

> *This is my first view. It contains information from both the IBBLOC and the CABLOC.*
> *The function code of the aborting trx is [**IBFCCD(1)**] and it was processed on PID [**CABCB(2)**].*

When the *view* is displayed, the values of IBFCCD(1) and CABCB(2) will replace ***[IBFCCD(1)]*** *and* ***[CABCB(2)]*** in the text and the *view* will looks like this:

> *This is my first view. It contains information from both the IBBLOC and the CABLOC.*
> *The function code of the aborted trx is **LN** and it was processed on PID **3821**.*

This is a very basic *view*, but you can make them as complex and informative as you wish. The ***[expression]*** field can have the following formats:

[***expression***]	Simple *expression.* The display format is dependent upon the value. Ascii, fieldata, octal or decimal is chosen based upon what appears to be the best choice.
[**F:*expression***]	Format-controlled *expression*. "**F**" is the display format and must take one of the following values: **A**: Ascii, **X**: Ascii overlays octal, **F**: Fieldata, **O**: Octal, **D**: Decimal
[**F:**ADDRESS(***expression***)]	ADDRESS function**.** This function will retrieve the value of the *address* given by the result of the *expression*.
[**F:**BITS(***expr1,expr2,expr3***)]	BITS function. This function will return the value of the specified bits. It is identical to the Fortran BITS function.
[**F**:REG(X3)]	REG function. This function will return the value of the specified AXR$ register and Extended Mode B-registers.
[**F**:DBANAME(***expression***)]	DBANAME function. This function will return the ascii name of the requested dba number provided by the *expression*.
[**F**:PDBNAME(***expression***)]	PDBNAME function. This function will return the ascii name of the requested pdb number provided by the *expression*.
[**F**:PRGNAME(***expression***)]	PRGNAME function. This function will return the FD name of the program with the provided lib/bank in the *expression*.

The expression itself must be in the format described in the Expression Window section.

Special system-related *expressions* can also be used within a *view*. You can see these by selecting the "**Show View values"** in the Pop-up menu of the **Expression** window.

Auto-generation of *views*

Views can be auto-generated from **$F** or **–F** procs. If a **$F** proc is available, it should be used as it typically contains the documentation lines. However, if a **$F** does not exist and the **–F** proc contains documentation lines, it then should be used for auto-generating the *view.*

You can auto-generate a *view* from a proc by using the batch program **SEDGEN/BATCH**. This program has a variety of functions with *view* construction being one. After copying **SEDGEN/BATCH** from SYSABS into TPF$, just execute it with the following input cards:

>@**XQT SEDGEN**
>PID number as requested
>Airline Code as requested
>**VIEW**
>**VIEW *view-name [apl]COM.proc$f***

An auto-generated *view* that is named ***view-name*** will be created and placed in TPF$. Generally, the convention of naming a view by preceding the proc name with "**V-**" is used. For example, CA$F will become V-CA$F and IV-F will become V-IV-F.

Sometimes auto-generated *views* need editing before they function properly. This is because the indexes for defines are set to one (1) by default when generating the *view* (e.g., [MYDEF(1,1)]). Some defines expect a different index value and thus these ***expressions*** must be edited to use the correct value.

When you are satisfied with a *view*, you can make it public to all users of The Visual Dump Analyzer by copying it into the file **SED$VIEW**. All *views* within this file are automatically shown in the "**View**:" drop-down box of the **Data selection** toolbar. Be aware that the **SED$VIEW** file exists on all systems running The Visual Dump Analyzer so public views should be copied onto all of these systems.

If you wish to have a *view* for your own use, you can copy it into your personal file and then read the private *view* by clicking the **View:** button on the **Data selection** toolbar. In the dialog box provided, just enter its name and path, and it will be included in the **View:** drop-down menu.

If your *view* contains symbols from a new **$F** or **–F** proc, you will need to create a **Resolve Program** in order to *teach* The Visual Dump Analyzer how to resolve your new ***expression*** names. (See ***Resolve Programs*** section)

The VDA Knowledge Database

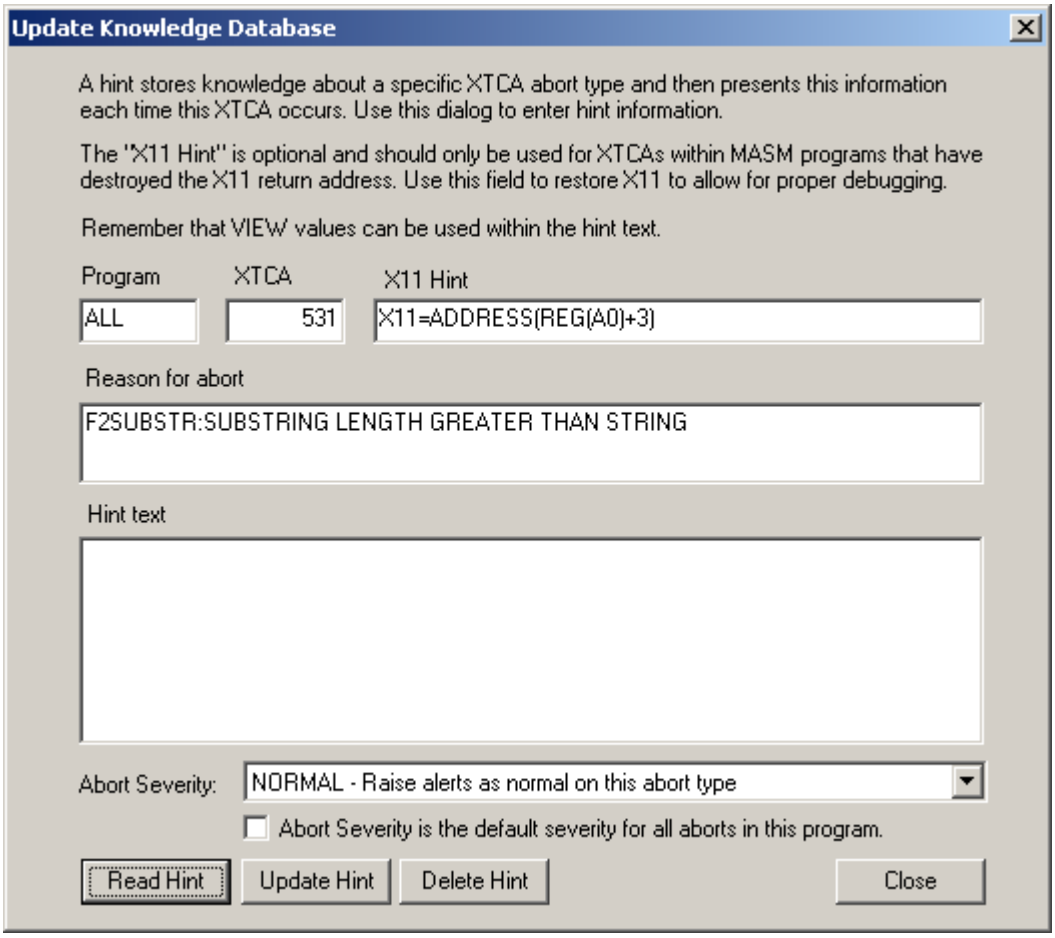
The Visual Dump Analyzer's Knowledge Database allows you to save hints and help information related to a specific abort code and program. These hints, which can contain ***expressions*** making for very thorough explanations, are automatically displayed the next time an abort occurs that has the identical conditions. These hints will be displayed in the **General Info** window of The Visual Dump Analyzer.

Saving this valuable information makes analyzing future dumps of this type more accurate and effective. Further, this knowledge is automatically shared among all programmers – a benefit especially helpful to those that are less experienced. And finally, as The Visual Dump Analyzer matures, many dumps will have helpful notes readily available so the overall process of dump analysis will be more efficient.

Hints can be thought of as small *views* for their constructs are the same: freeform text and ***expressions***. You can update the VDA Knowledge Database by clicking "**Update Hint**" on the Pop-up menu of the **General Info** window.

Hints are associated with dumps by their program name and XTCA error code. Unique XTCA errors produced by ACB routines should use **ALL** as the program name for they will be valid for all programs. Hints logged with a program name of **ALL** will be displayed whenever its related error code occurs throughout the system.

The VDA Knowledge Database contains three types of information: the "**Reason for abort**", "**Hint text**" and "**X11 Hint**".



When you are analyzing a dump and you open this dialog, the hints that are associated with the dump will be automatically displayed in the prompt areas for easy updating.

If you wish to update a hint within the database, just enter the program name and XTCA error code; and press "**Read Hint**". After you have added or changed help information within the dialog box, press "**Update Hint**" and the data will be stored in the VDA Knowledge Database on all mainframes. To delete a hint, retrieve it and press "**Delete Hint**". This will remove it from all systems.

Reason for abort text

This abort text is a general explanation of the abort. Being composed of freeform text, it resides in a separate record of the database from the "**Hint text**". If the "**Reason for abort**" relates to an ACB routine and it is thus valid for all programs, the syntax allows you to include the name of the program that produces this XTCA call followed by a colon as the first part of the text. (e.g., SFXDBA:text). This provides The Visual Dump Analyzer with the ability to obtain the source code associated with this abort, if needed.

Hint text

The **Hint text** is composed of freeform text and ***expressions***. By using ***expressions***, you can create very informative hints by incorporating values from the dump so they will be immediately available in the **General Info** window the next time an abort of this type occurs. Moreover, as The Visual Dump Analyzer matures, these hints will become more detailed resulting in greater efficiency with dump analysis in the future. In order to preserve the format you enter, line breaks are kept within the text.

X11 Hint text

The "**X11 Hint**" is a specialized hint used only for MASM subroutines – usually within an ACB – that destroy or reused the X11 (return) register. When these MASM subroutines abort, it can be difficult to find out where these subroutines are called from.

Typically, these routines store the X11 register in a *save* location for debug purposes. Thus, the "**X11 Hint**" explains how to regain the X11 register from the *save* location. This allows The Visual Dump Analyzer to calculate the correct return and, thus, be able to display the source line that actually called this routine.

The format of the "**X11 Hint**" is as follows:

X11=<*expression*>

The following is an example of when an X11 Hint is very helpful. The substring routine, F2SUBSTR is called when partial strings are manipulated in Fortran (e.g., LINE = WORKLN(1:12)). When a 531 error occurs indicating the substring length is greater than the string, F2SUBSTR saves the original value of the X11 register in IBNOTE(1) and then calls XTCA.

In order to determine which Fortran statement actually has a substring error, a programmer would likely have to spend considerable time studying the system software before it is recognized that IBNOTE(1) actually holds the X11 return address. Then, the absolute address would need to be converted to the actual line number where the substring error occurs - a tedious and time-consuming process.

By simply entering **X11=IBNOTE(1)** as the "**X11 Hint**" for error 531 with **ALL** programs, this extensive analysis will forever be avoided. Moreover, by entering this hint *once* in the database, aborts of this type will always be able to be solved quickly and accurately.

Abort Severity

The Abort Severity setting enables you to mark specific abort types having a severity that is either higher or lower than normal. The severity setting is used in two ways. First, it controls when Alert Messages are raised and, second, it determines when application programmers should be notified of the aborts when the alert is raised outside normal working hours.

SAFE	Never alert on this abort type
LOW	Postpone alerts outside working hours until next working day
MINOR	Postpone alerts during the night until next morning
NORMAL	Raise alerts as normal on this abort type
HIGH	Alert on ALL aborts of this type

The SAFE severity level is one of two severity settings that are used to control when Alert messages are raised. Aborts types marked SAFE will *never* cause an Alert without regard to how the Alert Message Warning Levels are configured. Also, these aborts are not counted when checking the number of aborts against the Warning Levels. Dumps with a SAFE severity will be colored GREEN in the aborts reports to inform that these aborts are considered to not be harmful.

Use the SAFE setting when you are not interested in an alert due to this type of abort. This level can also be used when an aborting program has been fixed, but for it is not be placed into production until a later time. In this case, it may be desirable to temporarily mark the abort type as SAFE until the program is moved into production.

The HIGH severity setting is the second severity setting that is used to control when Alert messages are raised. Aborts types with a HIGH severity setting will *always* cause an Alert without regard to how the Alert Message Warning Levels are configured. Use the HIGH severity setting with abort types that you always want to raise an Alert message so you can take immediate action. Dumps with a HIGH severity setting will be colored RED in the abort reports to inform that they are of HIGH severity.

The LOW and MINOR severity settings do not control when Alert messages are raised. These severity settings instead instruct the system operators to delay a call to the responsible programmers in the Alert Message text. Note that the Alert will be raised, but the text in the Alert message will request the notification to the application programmers to be delayed until the next morning or Monday morning according to the severity setting.

Alerts messages that are due to aborts with a severity setting of MINOR will request the notification of the responsible programmer to be delayed until next morning if it occurs between 9PM and 9AM, whereas it will request immediate notification if it occurs outside this time period.

Similarly, Alert messages that due to aborts with a severity setting of LOW will request notification of the responsible application programmer to be delayed until next morning if it occurs after 4PM on Monday through Thursday. If it occurs after 4PM Friday or during the weekend, it will request notification to be delayed until Monday morning. Alerts during normal working hours (9AM through 4PM) will cause notification to be immediate.

The NORMAL severity setting is the default, and it is also the setting used for all abort types that are not updated in **The VDA Knowledge Database**. Alerts based on NORMAL severity aborts will request the notification of the application programmers to be immediate.

Hints logged with a program name of **ALL** can – for safety reasons – be set to severity NORNAL only. You can, however, always log a hint for a specific program with the same XTCA error code and then change its severity setting.

Because the severity setting for aborts can have a significant impact on the way the system is being monitored, only known users (See the ***User Identification***) can change the severity settings from NORMAL.

Resolve Programs

A **Resolve Program** is an auto-generated FTN program that is used by The Visual Dump Analyzer to resolve ***expressions*** that relate to one or more application procs. The following program fragment gives you an idea of its simple, but effective structure.

```
IF (exp.EQ.'IBFUNC') THEN
    VALUE = IBFUNC(PAR1)
ELSE IF (exp.EQ.'IBLNSZ') THEN
    VALUE = IBLNSZ(PAR1)
ELSE IF (exp.EQ.'IBFCCD') THEN
    VALUE = IBFCCD(PAR1)
ELSE IF ...
```

This logic ensures the actual FTN defines are used when an ***expression*** is being resolved.

Once a Resolve Program is generated, it should be treated as any other application program. Further, when a proc is updated, the associated Resolve Program must be compiled and mapped. Finally, when defines are added or removed from a proc, the associated Resolve Program must be regenerated.

In order to generate a Resolve Program from a proc, all define names must be prefixed with the proc mnemonic. (e.g., all defines within IP$F must being with "IP") Furthermore, the proc must be a standard USAS proc that has the xxFSPEC and xxFDEF include areas.

Defining Resolve Programs

Defining a Resolve Program is a two-step process. The first step informs The Visual Dump Analyzer of which procs are to be handled by the Resolve Program, whereas the second step generates the source code proc (SEDUSR-F) that enables The Visual Dump Analyzer to call the Resolve Program.

Step 1 – Associating procs to Resolve Programs

This step defines the Resolve Program itself and associates one or more procs to it. After copying **SEDGEN/BATCH** from SYSABS into TPF$, simply execute it with the following input:

>@**XQT SEDGEN**
>PID number as requested
>Airline Code as requested
>**INSERT**
>**INSERT** ***program-name [apl]COM.proc-f***

This will associate the ***[apl]COM.proc-f*** to the Resolve Program named ***program-name*** and ensure that this Resolve Program will be able to resolve all the ***expression*** names (defines) from the [**apl**]**COM.proc-f**. The procs supplied here should be from the application *COM* files, because the proc will be included in the Resolve Program using [**apl**]LIB$.**proc**FSPEC and [**apl**]LIB$.**proc**FDEF statements.

Multiple procs can be associated with a single Resolve Program. To do this, just enter the same ***program-name*** as the input to SEDGEN for each proc that you wish to be contained within it.

Step 2 – Updating the SEDUSR-F proc

This step will update the system proc SEDUSR-F. This proc is used on the mainframe by The Visual Dump Analyzer to associate expression names with their associated Resolve Programs. Because of the global affect of this step, performing it should be left to the person responsible for supporting The Visual Dump Analyzer.

After copying **SEDGEN/BATCH** from SYSABS into TPF$, the SEDUSR-F proc is updated in the following manner:

>@XQT SEDGEN
>PID number as requested
>Airline Code as requested
>**NAME**
>**NAME SEDUSR-F**

The updated version of **SEDUSR-F** will be placed in TPF$ and must be compiled into The Visual Dump Analyzer program **SEDVAL**. This process should be performed on a regular basis in order to simplify the generation of application Resolve Programs.

Generating Resolve Programs

Once a Resolve Program is defined, it can be generated. When generating a Resolve Program, all the procs associated with it are scanned and code is produced to ensure that all defines within the procs can be handled. Therefore Resolve Programs must be re-generated every time defines are added or removed from the procs they are associated with. Failure to do so will cause compile errors when the Resolve Program refers to defines that no longer exist. Further, The Visual Dump Analyzer will be unable to resolve new defines when they are introduced.

Because The Visual Dump Analyzer cross-references the Resolve Programs and their associated procs, generating a Resolve Program is very easy. After copying SEDGEN/BATCH from SYSABS into TPF$, you only need to supply the Resolve Program name in the following manner:

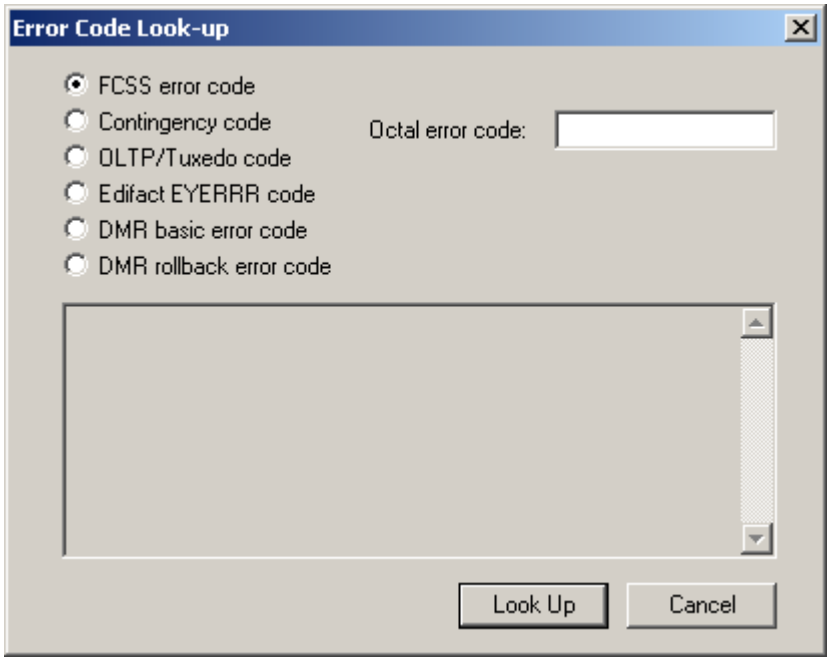
>**@XQT SEDGEN**
>PID number as requested
>Airline Code as requested
>**NAME**
>**NAME *program-name***

The generated the Resolve Program will be placed in TPF$. If one or more of the procs associated with this Resolve Program contain defines that need other procs to be included, these can be added to the SGS cards stored in the element SED$FILE.SED$BLD. Please refer to this element for a description of this process.

Once the Resolve Program is ready to be introduced into the system, use the same procedures as those used with other application programs. This will result in it being compiled, mapped, and copied into a library/bank.

Error Code Look-up

The Visual Dump Analyzer is aware of all contingency (including HVTIP), FCSS, OLTP (Tuxedo), Edifact input and output handler (EYERRR) error codes and DMS error codes. The descriptions for these error codes are not only used during dump analysis, but are also available to you upon request. You can reference these error codes by entering "**Tools | Error Codes**".



Simply enter the error code and press "**Look Up**".

The Visual Dump Analyzer Log File

The Visual Dump Analyzer has a log file named Vda.log. This log file of ascii text format can be used to trace the status of Tuxedo communication with the mainframe. Enter "**Tools | Settings**" and then select the **General** "**Settings**" tab to toggle the logging of data to this file. With the default setting being OFF, you should only change this if you are experiencing communication problems.

Fortran Options

The Visual Dump Analyzer is dependent on the compilers building diagnostic information in the absolutes for the purpose of assisting with debugging. The compilers must be requested to do so by supplying specific options on the compile statement. Failure to generate the diagnostic tables will not prevent The Visual Dump Analyzer from working with an absolute or a dump caused by it, however, functions specifically related to variables will not be available.

Basic Mode Fortran Options

Diagnostic tables under location counter 3 are used by The Visual Dump Analyzer to determine the structures within an absolute element. These tables are created when an **F-option** is included on the compiler's processor call (i.e., @FTN,F). Therefore, in order for The Visual Dump Analyzer to function, it is critical that the F-option be used when compiling programs. These diagnostic tables will not be part of the executable code, and thus they will not increase an absolute's size in the HVTIP online file.

The **Z-option** will force the ASCII Fortran compiler to optimize the generated MASM code for efficiency. This may result in instructions generated by one FTN source line to be mixed with code generated by surrounding source lines. Thus, when viewing MASM code using "**Show MASM Code**", The Visual Dump Analyzer will display MASM instructions by individual source lines, but be aware that some instructions may actually be generated by source lines that are nearby.

Extended Mode UCS options

To inform the UCS compilers to generate the SDD (Symbolic Debugging Dictionary) diagnostic tables, you must specify the keyword option "DEBUG/MONITOR" on the compiler statement. As with Basic Mode, this will not influence the actual generated code, but only ensure that SDD banks are created. UCS programs have several levels of SDD information, with "DEBUG/MONITOR" being the highest and desired level for The Visual Dump Analyzer.

If you specify the "OPTIM" keyword option rather than the "NO-OPTIM" keyword option, the UCS compilers will optimize the generated code for speed. The Visual Dump Analyzer is able to handle this optimized code, but be aware that this can cause the instructions generated by one source line to be mixed with code generated by the surrounding source lines.

Monitoring USAS Applications

The Visual Dump Analyzer keeps track of all aborts that occur on a system by recording them in The Visual Dump Analyzer's common bank (SED$Q). This common bank has several purposes. First, it can provide the user with a quick *System Abort Status* or *Abort Report* by presenting an overview report of all aborts that have occurred on the system during the last hours, days or weeks.

Second, the Abort Report can be used to monitor all USAS applications on a given system and automatically raise an Alert Message if it detects a pattern to the aborts, or if the number of aborts simply grows too large within a specific time frame.

Third, the Abort Report enables the user to quickly find and retrieve dumps.

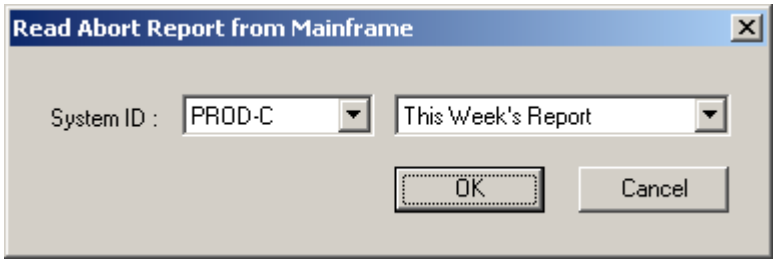
Fourth, The Visual Dump Analyzer can be used to control the number of dumps that are actually processed (by the USAS SYS contingency handler) when many identical aborts occur on the system within a short period of time. (See ***The Visual Dump Analyzer Basic Mode - DABT function)***.

Abort Reports

The Abort Report contains one line of information for each abort that has occurred on the system being monitored. Besides date and time stamps, dump name, function code, pid number and application names the report includes a wide range of diagnostic information. Abort Reports will contain much more information than the Basic Mode equivalent – the DABT function – as it is not restricted to 80 characters per line. Further, it gives the user control of what is to be displayed in the report.

Reading Abort Reports from the mainframe

To read an Abort Report from the mainframe, select "**File | New Abort Report**" or click the "**Abort Report**" icon on the main menu. In the dialog, you can select the desired system along with the time period covered by the Abort Report.



The Abort Report will retrieve information on all dumps from the time of retrieval for a specified time period back in time. The time period covered by the report is indicated by the report name - i.e., "**Today's Report**", "**Today's and Yesterday's Report**", "**This Week's Report**" or "**Complete Report**".

Once the report has been retrieved, the data displayed in the report can be limited by use of the "**Report Filter**".

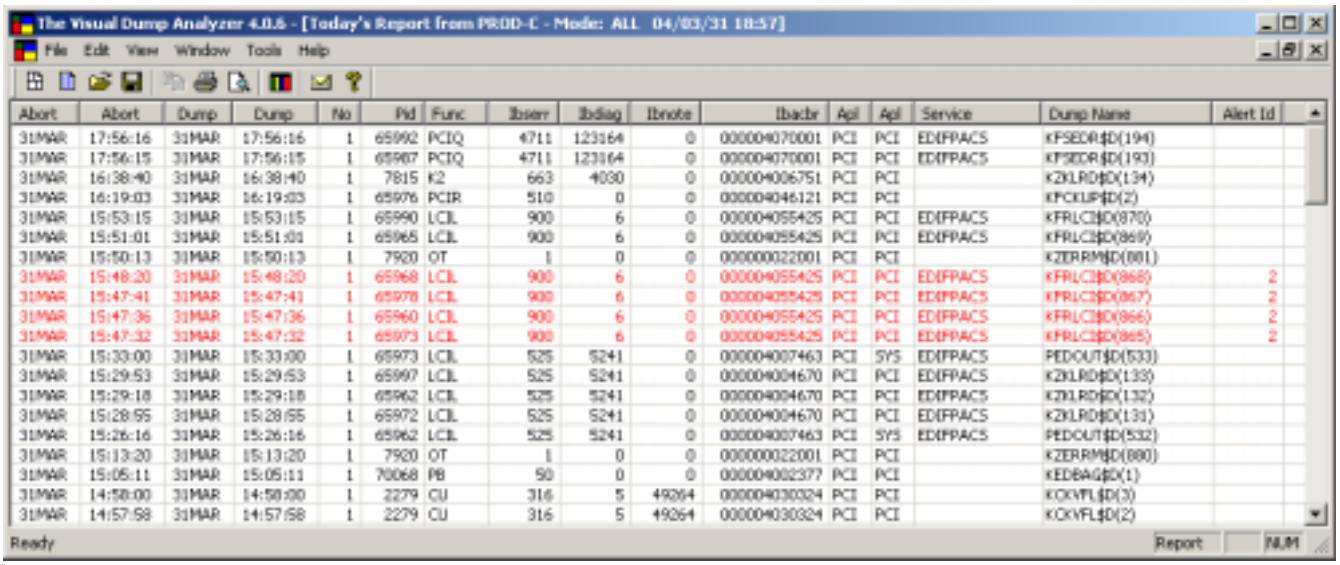
Abort Report Types

Once an Abort Report has been retrieved from the mainframe, it can be shown using one of three report forms. Each of these reports will present the information differently, and the display between these reports can be toggled back and forth using the Abort Report Pop-up menu.

Showing Abort Reports

This report form – which is the default - includes one line for each abort along with all its diagnostic information. Being sorted with the newest abort on the first line, the Abort Report presents you with an immediate and accurate status on how the USAS applications are functioning. Further, by combining this report with the "**Auto**

Refresh" and **Alert Messages**, you are provided a powerful tool for monitoring the USAS applications and alerting you immediately when aborts occur. By detecting aborts very early and raising alerts, The Visual Dump Analyzer enables you to take action before the aborts become a serious problem.

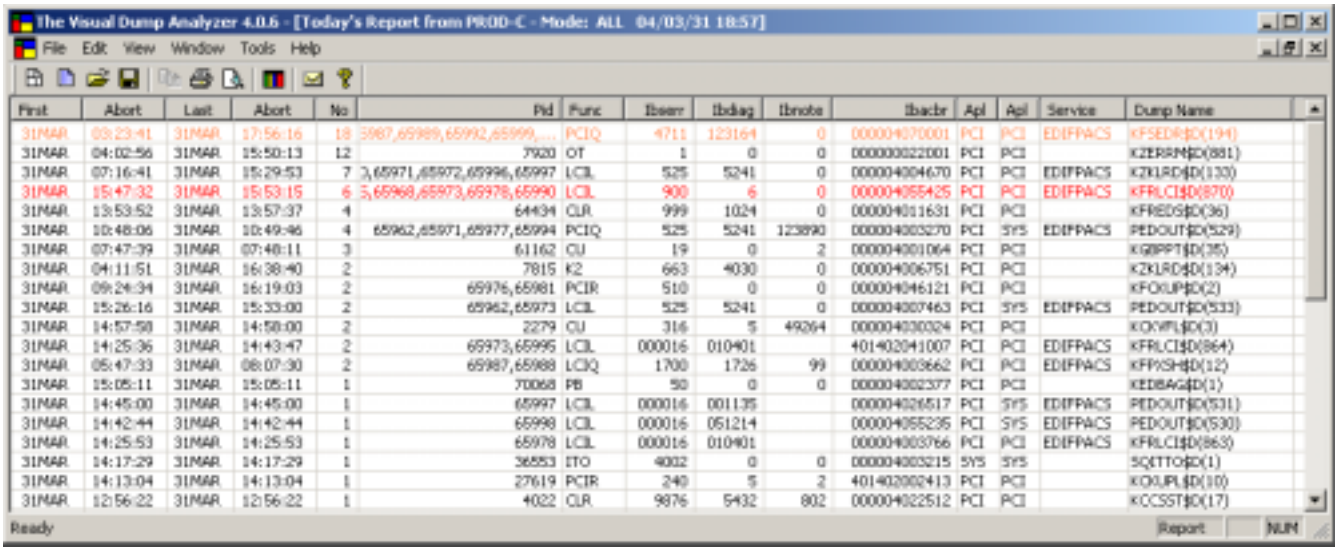


Abort	Abort	Dump	Dump	No	Pid	Func	Ibserr	Ibdiag	Ibnote	Ibacbr	Apl	Apl	Service	Dump Name	Alert Id
31MAR	17:56:16	31MAR	17:56:16	1	65992	PCIQ	4711	123164	0	000004070001	PCI	PCI	EDIFPACS	KFSEDR$D(194)	
31MAR	17:56:15	31MAR	17:56:15	1	65987	PCIQ	4711	123164	0	000004070001	PCI	PCI	EDIFPACS	KFSEDR$D(193)	
31MAR	16:38:40	31MAR	16:38:40	1	7815	K2	663	4030	0	000004006751	PCI	PCI		KZKLRD$D(134)	
31MAR	16:19:03	31MAR	16:19:03	1	65976	PCIR	510	0	0	000004046121	PCI	PCI		KFOKUP$D(2)	
31MAR	15:53:15	31MAR	15:53:15	1	65990	LCIL	900	6	0	000004055425	PCI	PCI	EDIFPACS	KFRLCI$D(870)	
31MAR	15:51:01	31MAR	15:51:01	1	65965	LCIL	900	6	0	000004055425	PCI	PCI	EDIFPACS	KFRLCI$D(869)	
31MAR	15:50:13	31MAR	15:50:13	1	7920	OT	1	0	0	000000022001	PCI	PCI		KZERRM$D(881)	
31MAR	15:48:20	31MAR	15:48:20	1	65968	LCIL	900	6	0	000004055425	PCI	PCI	EDIFPACS	KFRLCI$D(868)	2
31MAR	15:47:41	31MAR	15:47:41	1	65978	LCIL	900	6	0	000004055425	PCI	PCI	EDIFPACS	KFRLCI$D(867)	2
31MAR	15:47:36	31MAR	15:47:36	1	65960	LCIL	900	6	0	000004055425	PCI	PCI	EDIFPACS	KFRLCI$D(866)	2
31MAR	15:47:32	31MAR	15:47:32	1	65973	LCIL	900	6	0	000004055425	PCI	PCI	EDIFPACS	KFRLCI$D(865)	2
31MAR	15:33:00	31MAR	15:33:00	1	65973	LCIL	525	5241	0	000004007463	PCI	SYS	EDIFPACS	PEDOUT$D(533)	
31MAR	15:29:53	31MAR	15:29:53	1	65997	LCIL	525	5241	0	000004004670	PCI	PCI	EDIFPACS	KZKLRD$D(133)	
31MAR	15:29:18	31MAR	15:29:18	1	65962	LCIL	525	5241	0	000004004670	PCI	PCI	EDIFPACS	KZKLRD$D(132)	
31MAR	15:28:55	31MAR	15:28:55	1	65972	LCIL	525	5241	0	000004004670	PCI	PCI	EDIFPACS	KZKLRD$D(131)	
31MAR	15:26:16	31MAR	15:26:16	1	65962	LCIL	525	5241	0	000004007463	PCI	SYS	EDIFPACS	PEDOUT$D(532)	
31MAR	15:13:20	31MAR	15:13:20	1	7920	OT	1	0	0	000000022001	PCI	PCI		KZERRM$D(880)	
31MAR	15:05:11	31MAR	15:05:11	1	70068	PB	50	0	0	000004002377	PCI	PCI		KEDBAG$D(1)	
31MAR	14:58:00	31MAR	14:58:00	1	2279	CU	316	5	49264	000004030324	PCI	PCI		KOKVFL$D(3)	
31MAR	14:57:58	31MAR	14:57:58	1	2279	CU	316	5	49264	000004030324	PCI	PCI		KOKVFL$D(2)	

By combining the Abort Reports with "**Report Filters**", you can use The Visual Dump Analyzer to monitor specific programs or functions. Especially in cases where modified programs are placed on a test system before eventually moved to the production environment, you may wish to use The Visual Dump Analyzer to ensure you are informed on every abort taken by these programs.

Showing Abort Totals

The Abort Totals Report provides an overview of how many different types of aborts have occurred during the previous days. In the Abort Totals Report, each line shows the number of occurrences that have occurred for a specific abort type. The Abort Totals Report is initially displayed with the abort type having the most occurrences on the top line. This report gives you a quick overview of what is the most common abort or immediate problem allowing you to direct your attention to where it is needed most.



First	Abort	Last	Abort	No	Pid	Func	Ibserr	Ibdiag	Ibnote	Ibacbr	Apl	Apl	Service	Dump Name
31MAR	03:23:41	31MAR	17:56:16	18	5987,65989,65992,65999,...	PCIQ	4711	123164	0	000004070001	PCI	PCI	EDIFPACS	KFSEDR$D(194)
31MAR	04:02:56	31MAR	15:50:13	12	7920	OT	1	0	0	000000022001	PCI	PCI		KZERRM$D(881)
31MAR	07:16:41	31MAR	15:29:53	7	3,65971,65972,65996,65997	LCIL	525	5241	0	000004004670	PCI	PCI	EDIFPACS	KZKLRD$D(133)
31MAR	15:47:32	31MAR	15:53:15	6	5,65968,65973,65978,65990	LCIL	900	6	0	000004055425	PCI	PCI	EDIFPACS	KFRLCI$D(870)
31MAR	13:53:52	31MAR	13:57:37	4	64434	CLR	999	1024	0	000004011631	PCI	PCI		KFREDS$D(36)
31MAR	10:48:06	31MAR	10:49:46	4	65962,65971,65977,65994	PCIQ	525	5241	123890	000004003270	PCI	SYS	EDIFPACS	PEDOUT$D(529)
31MAR	07:47:39	31MAR	07:48:11	3	61162	CU	19	0	2	000004001064	PCI	PCI		KGBPPT$D(35)
31MAR	04:11:51	31MAR	16:38:40	2	7815	K2	663	4030	0	000004006751	PCI	PCI		KZKLRD$D(134)
31MAR	09:24:34	31MAR	16:19:03	2	65976,65981	PCIR	510	0	0	000004046121	PCI	PCI		KFOKUP$D(2)
31MAR	15:26:16	31MAR	15:33:00	2	65962,65973	LCIL	525	5241	0	000004007463	PCI	SYS	EDIFPACS	PEDOUT$D(533)
31MAR	14:57:58	31MAR	14:58:00	2	2279	CU	316	5	49264	000004030324	PCI	PCI		KOKVFL$D(3)
31MAR	14:25:36	31MAR	14:43:47	2	65973,65995	LCIL	000016	010401		401402041007	PCI	PCI	EDIFPACS	KFRLCI$D(864)
31MAR	05:47:33	31MAR	08:07:30	2	65987,65988	LCIQ	1700	1726	99	000004003662	PCI	PCI	EDIFPACS	KFPXSH$D(12)
31MAR	15:05:11	31MAR	15:05:11	1	70068	PB	50	0	0	000004002377	PCI	PCI		KEDBAG$D(1)
31MAR	14:45:00	31MAR	14:45:00	1	65997	LCIL	000016	001135		000004026517	PCI	SYS	EDIFPACS	PEDOUT$D(531)
31MAR	14:42:44	31MAR	14:42:44	1	65998	LCIL	000016	051214		000004055235	PCI	SYS	EDIFPACS	PEDOUT$D(530)
31MAR	14:25:53	31MAR	14:25:53	1	65978	LCIL	000016	010401		000004003766	PCI	PCI	EDIFPACS	KFRLCI$D(863)
31MAR	14:17:29	31MAR	14:17:29	1	36853	ITO	4002	0	0	000004003215	SYS	SYS		SQITTO$D(1)
31MAR	14:13:04	31MAR	14:13:04	1	27619	PCIR	240	5	2	401402002413	PCI	PCI		KOKUPL$D(10)
31MAR	12:56:22	31MAR	12:56:22	1	4022	CLR	9876	5432	802	000004022512	PCI	PCI		KCCSST$D(17)

Classifying Aborts as Identical or Similar

Aborts are determined to be identical when they occur in the same program have identical diagnostic information. For XTCA aborts, the diagnostic information is considered identical when IBSERR, IBDIAG, IBNOTE and IBACBR are alike - whereas for XCONT aborts the contingency code and the contingency address need to be the same.

Aborts are considered to be similar when they occur in the same program and have the same IBSERR or have caused the same contingency.

Showing Abort Severity Report

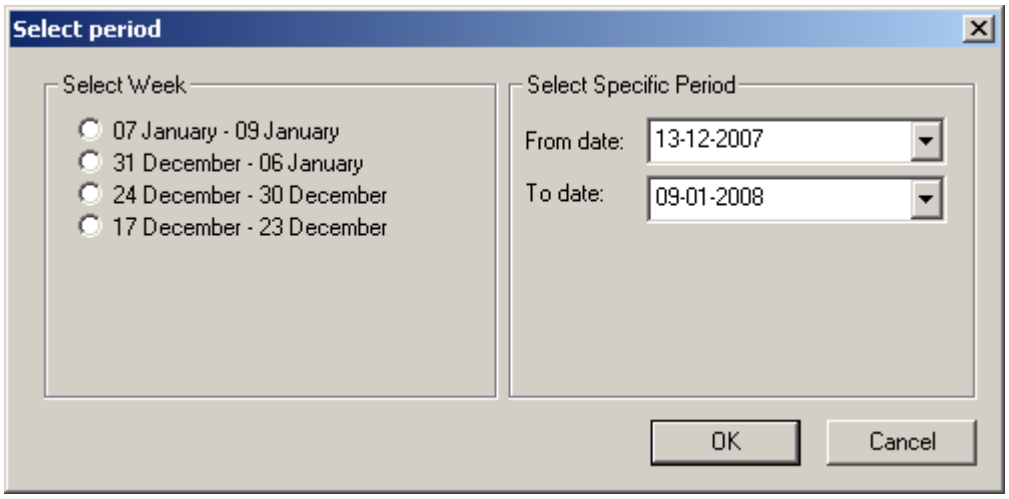
The Abort Severity Report shows all of the abort types that have been marked with an Abort Severity different than NORMAL. The Alert Severity Report is therefore a view into the Knowledge Database (see ***The VDA Knowledge Database*** later in this document), rather than into the SED$Q common bank. All the abort types that have been marked with a Severity other than NORMAL in the VDA Knowledge Database will appear in the Abort Severity Report – even though an abort of the specific type has never occurred.

As with all report types, double-clicking the IBSERR column will automatically cause The Visual Dump Analyzer to find and read the specific abort hint in the Knowledge Database and present it in the **Update Knowledge Database** dialog for inspection or change.

Showing Statistical Report

The Statistical Report will provide an overview for a given period of how many times each abort has occurred, what severity each abort has, and how often each application programmer has been called on duty due to Alert Messages raised by The Visual Dump Analyzer. It is organized by application and Abort Severity.

To display the Statistical Report, open a normal Abort Report, then right-click on it to open the Pop-up menu and finally select "**Show Statistical Report**". This will open the below dialog where you can select the period you wish to be covered by the statistics.



Once the period has been selected, The Visual Dump Analyzer will retrieve all Alert Messages raised on the SPO from the VDA SPO Agent (VdaSpo.exe). The VDA SPO Agent will return ALL Alert Messages back to The Visual Dump Analyzer GUI so it can create a complete Statistical Report that contains ALL messages – even those that are not raised from this instance of The Visual Dump Analyzer.

Once All Alert Messages have been retrieved for the selected period, The Visual Dump Analyzer will organize them according to USAS application and Abort Severity. Further it will show how many of the Alert Messages occurred during normal working hours, off working hours and during weekends.

Complete Report from PROD-E - Mode: ALL 08/01/09 15:33

ALL	Aborts	Pct	Types	Pct	Alerts	Pct	Off Hours	Pct	Weekend	Pct
SAFE	35	9	12	13	0	0	0	0	0	0
LOW	87	22	8	9	3	42	1	100	0	0
MINOR	0	0	0	0	0	0	0	0	0	0
NORMAL	265	68	67	77	4	57	0	0	2	100
HIGH	0	0	0	0	0	0	0	0	0	0
Total	387	100	87	100	7	100	1	100	2	100
RES	Aborts	Pct	Types	Pct	Alerts	Pct	Off Hours	Pct	Weekend	Pct
SAFE	29	7	10	11	0	0	0	0	0	0
LOW	21	5	5	5	1	14	0	0	0	0
MINOR	0	0	0	0	0	0	0	0	0	0
NORMAL	118	30	34	39	0	0	0	0	0	0
HIGH	0	0	0	0	0	0	0	0	0	0
Total	168	43	49	56	1	14	0	0	0	0
SYS	Aborts	Pct	Types	Pct	Alerts	Pct	Off Hours	Pct	Weekend	Pct
SAFE	5	1	1	1	0	0	0	0	0	0
LOW	0	0	0	0	0	0	0	0	0	0
MINOR	0	0	0	0	0	0	0	0	0	0
NORMAL	31	8	7	8	1	14	0	0	1	50
HIGH	0	0	0	0	0	0	0	0	0	0
Total	36	9	8	9	1	14	0	0	1	50
TKT	Aborts	Pct	Types	Pct	Alerts	Pct	Off Hours	Pct	Weekend	Pct
SAFE	0	0	0	0	0	0	0	0	0	0
LOW	1	0	1	1	0	0	0	0	0	0
MINOR	0	0	0	0	0	0	0	0	0	0
NORMAL	47	12	15	17	1	14	0	0	1	50
HIGH	0	0	0	0	0	0	0	0	0	0
Total	48	12	16	18	1	14	0	0	1	50

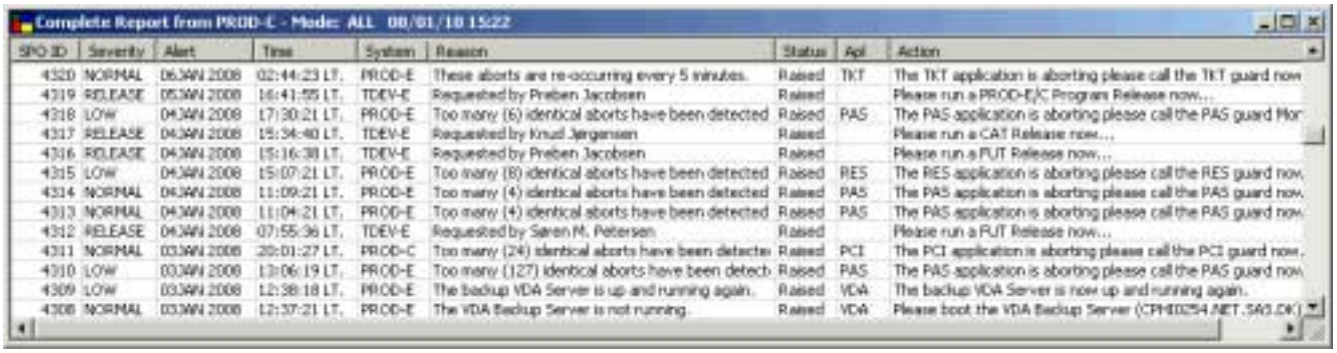
By displaying the number of times USAS application people have been called on-duty outside of normal working hours and during weekends, The Visual Dump Analyzer provides a method of keeping track of how often aborts within the USAS applications cause application people that are not on-site to be called in.

By organizing the aborts into different types (See ***Classifying Aborts as Identical or Similar***), these statistics can also provide guidance with the number of times the same USAS abort has caused problems and how often application people have being called on-duty due to “new” aborts. (See ***Raising Alert Messages on the SPO***)

Showing SPO Alerts

The SPO Alerts Report shows a history of all the Alert Messages raised on the Unisys operator Single Point of Operation (SPO) console. Even though the SPO Alert Messages were not raised from this instance of The Visual Dump Analyzer, ALL the SPO messages raised during the period covered by the Abort Report will be retrieved from the VDA SPO Agent (VdaSpo.exe).

The VDA SPO Agent is responsible for actually raising the Alert Messages as alarms on the SPO. While doing this, it keeps a history of all messages that were raised while also allowing them to be fed back to the Visual Dump Analyzer GUI and be shown in this SPO Alert report.



Complete Report from PROD-E - Mode: ALL 08/01/10 15:22

SPO ID	Severity	Alert	Time	System	Reason	Status	Apl	Action
4320	NORMAL	06JAN 2008	02:44:23 LT.	PROD-E	These aborts are re-occurring every 5 minutes.	Raised	TKT	The TKT application is aborting please call the TKT guard now
4319	RELEASE	05JAN 2008	16:41:55 LT.	TDEV-E	Requested by Preben Jacobsen	Raised		Please run a PROD-E/C Program Release now...
4318	LOW	04JAN 2008	17:30:21 LT.	PROD-E	Too many (6) identical aborts have been detected	Raised	PAS	The PAS application is aborting please call the PAS guard Mor
4317	RELEASE	04JAN 2008	15:34:40 LT.	TDEV-E	Requested by Knud Jørgensen	Raised		Please run a CAT Release now...
4316	RELEASE	04JAN 2008	15:16:38 LT.	TDEV-E	Requested by Preben Jacobsen	Raised		Please run a PUT Release now...
4315	LOW	04JAN 2008	15:07:21 LT.	PROD-E	Too many (8) identical aborts have been detected	Raised	RES	The RES application is aborting please call the RES guard now.
4314	NORMAL	04JAN 2008	11:09:21 LT.	PROD-E	Too many (4) identical aborts have been detected	Raised	PAS	The PAS application is aborting please call the PAS guard now.
4313	NORMAL	04JAN 2008	11:04:21 LT.	PROD-E	Too many (4) identical aborts have been detected	Raised	PAS	The PAS application is aborting please call the PAS guard now.
4312	RELEASE	04JAN 2008	07:55:36 LT.	TDEV-E	Requested by Søren M. Petersen	Raised		Please run a PUT Release now...
4311	NORMAL	03JAN 2008	20:01:27 LT.	PROD-C	Too many (24) identical aborts have been detecte	Raised	PCI	The PCI application is aborting please call the PCI guard now.
4310	LOW	03JAN 2008	13:06:19 LT.	PROD-E	Too many (127) identical aborts have been detect	Raised	PAS	The PAS application is aborting please call the PAS guard now.
4309	LOW	03JAN 2008	12:38:18 LT.	PROD-E	The backup VDA Server is up and running again.	Raised	VDA	The backup VDA Server is now up and running again.
4308	NORMAL	03JAN 2008	12:37:21 LT.	PROD-E	The VDA Backup Server is not running.	Raised	VDA	Please boot the VDA Backup Server (CPH00254.NET.SAS.DK)

To view the entire text for an Alert Message, simply click the "**SPO ID**" in column 1 and a yellow Pop-up window will be displayed showing the entire Alert Message. To close this Pop-up window, just move the mouse outside the windows boundaries and it will close.

This report provides a quick and accurate overview of the Alert Messages raised as Alarms on the SPO during the period covered by the Abort Report you were looking at. (See ***Raising Alert Messages on the SPO***)

Customizing the Abort Report Layout

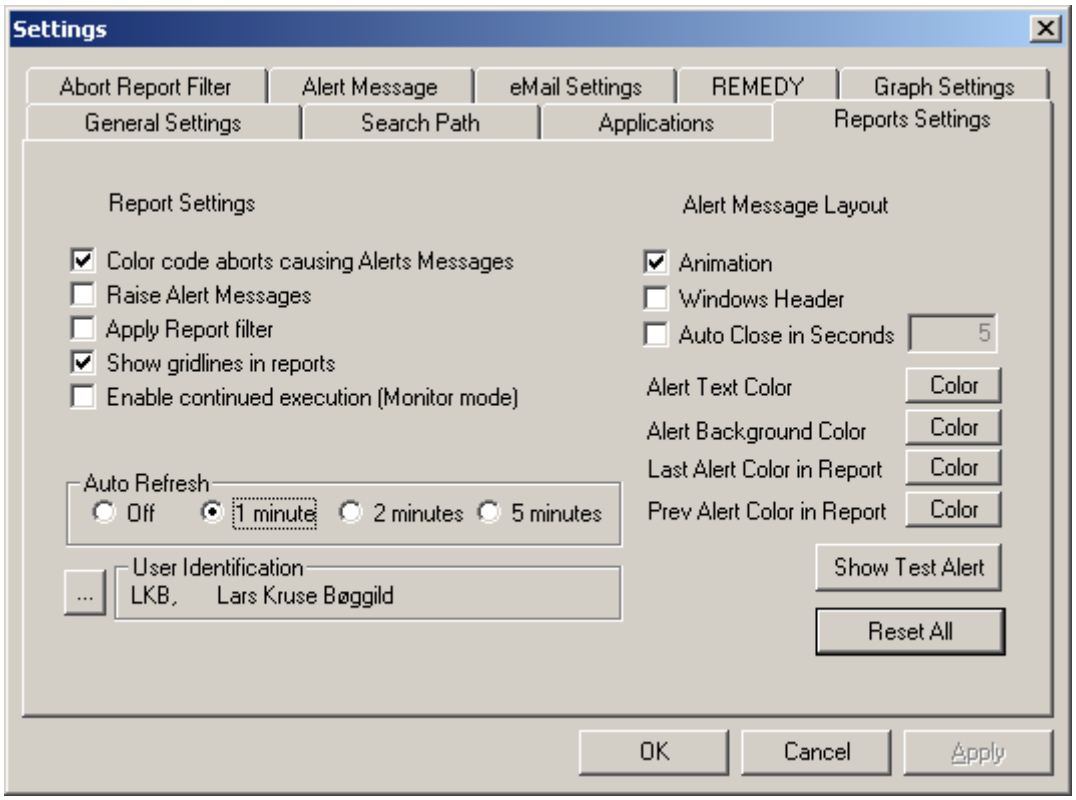
An Abort Report contains a wide range of information, some of which may not be of interest to you. By clicking the column header and selecting *Hide*, the column can be removed. The width and location of individual columns can also be changed by dragging the columns to their new location. Finally, the justification within the column can also be modified.

Once you have created the layout that you like, you can make it persistent by selecting "**Tools | Column Layout | Save Column Layout**". In this way you can create and save a layout for each of the 3 report types. To quickly set all the column widths relative to the data within them, just click on the "**Resize Column**" icon on the main menu.

To restore the default setting, select "**Tools | Column Layout | Reset Column Layout**".

Besides these settings, you can choose to automatically apply a preset "**Report Filter**" when the Abort Report is initially opened. This is done by selecting "**Tools | Settings | Report Settings**" and then checking the tic mark for "**Apply Report Filter**". In the dialog you can also ensure that the Abort Report is set to be continuously refreshed when it is first opened by setting the tic mark for "**Raise Alert Messages**". The settings in this dialog control the general behavior of the Abort Reports along with their initial settings once they are first opened. For an individual Abort Report, you can also chance these settings via its Pop-up menu.

In the "**Report Settings**" dialog you can customize the colors used to identify aborts in Abort Report that have caused **Alert Message** to be raised. By default the aborts that caused the last **Alert Message** will be colored **RED**, whereas the color used to identify previous **Alert Messages** are colored **ORANGE**.



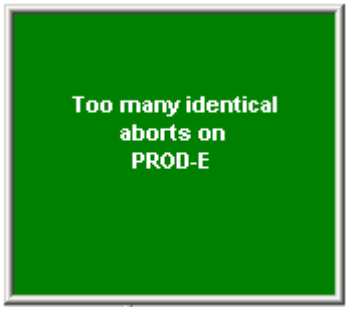
The layout and behavior of the alert message warning window itself can also be customized in this dialog. Use the "**Show Test Alert**" button to raise a test alert and verify the setting.

Auto refreshing Abort Reports

To ensure that the Abort Report is kept up to date The Visual Dump Analyzer periodically checks the monitored system for new aborts. In the Pop-up menu you can toggle the refresh time period between 1, 2 or 5 minutes - or you can turn the "**Auto Refresh**" logic completely OFF. If new aborts are found they are retrieved and merge into the report and the display is refreshed to reflect these new aborts. To manually refresh an Abort Report press **F5** and the report will be immediately refreshed, even if "**Auto Refresh**" logic is turned OFF.

Raising Alerts Messages

Raising **Alert Messages** is a powerful feature that will immediately inform you of when the monitored applications start creating aborts. Whenever the refresh logic finds and retrieves new aborts, the report is analyzed and checked to verify if these new aborts have caused the **Alert Message Warning levels** to be exceeded. If the "**Alert Me**" tic mark in the Pop-up menu is set and the analysis of the report reveals that one or more of the warning levels have been exceeded, an **Alert Message** is raised.



An **Alert Message** is raised by animating a small warning window in the bottom right corner of your screen. This window will be shown as the top-most window so that it will always stay visible. It will, however, not take the input focus so you can continue working without interruption.

The **Alert Message** will be raised even if The Visual Dump Analyzer application is not the active application at the time. Even if The Visual Dump Analyzer application is in the background or is minimized, the **Alert Message** will be raised.

Once the **Alert Message** is raised, you can use it to quickly activate The Visual Dump Analyzer application by clicking the warning text itself. The Visual Dump Analyzer will be brought to the foreground and maximized, and the Abort Report that has raised the **Alert Message** will likewise be maximized to give you the best overview. The aborts that have caused one of the **Alert Message Warning Levels** to be exceeded will be colored **RED** and the Alert Id column will be updated with the ID number of the **Alert Message**.

To see the **Alert Message Analysis** text giving the result of the analysis and explanation of the error, click the Alert Id in the Abort Report.

If the white space in the warning window is clicked rather that the text itself, the warning window will be closed and The Visual Dump Analyzer will not be activated.

Because the **Alert Message** logic is based on an analysis of the new aborts retrieved by the *refresh logic*, this feature must be turned ON. Turning the refresh logic OFF will therefore cause the **Alert Message** to be turned OFF as well.

Alert Message Analysis

The **Alert Message Analysis** gives the result of the Abort Report analysis. This summary gathers information from the dumps themselves, their diagnostic information, the **VDA Knowledge Database** and from the built-in knowledge of the USAS applications.

All this information is processed and the final result is used to generate a natural language text explaining as best possible the reason for the aborts. To see this analysis, click the *Alert Id* in the Abort Report and the analysis will be shown in a yellow Pop-up window. You can close the **Alert Message Analysis** window by simply moving the mouse outside its border.

Besides a short header line identifying the Alert Message and showing its status, the analysis text is divided into the below sections:

Action…
This section informs you which application or applications are responsible, along with whether this is a new **Alert Message** or an **Identical Alert Message** that has been raised in the past hour.

This section can also inform you of who should be called to inform them of these aborts. This information is intended for **Alert Messages** that are received by computer center operators rather than programmers.

Reason…
This section informs of what **Alert Message Warning Level** was exceeded and by how much.

Phone List…
The Phone List displays the responsible application programmers along with their phone numbers. The responsible programmers are found in the Program Book if an entry for the aborting program exists or, if none exists, the responsibility for the aborting program is assumed to be shared among all programmers working in the aborting application.

Analysis….
This section gives a natural language text describing the abort in as much detail as possible. The text will depend on the actual aborts and their origin. If the **VDA Knowledge Database** contains information for this abort type, it is included in the analysis.

```
VDA Application Alert 1 from TDEV-E (Not Raised)

Action...
The VDA application is aborting please call the VDA guard now.

SEDANA Phone list in call priority order...
LKB, Lars Kruse Baggild - Ext:25028 Mobile:no Home:3294 3194

Reason...
Too many (5) identical aborts have been detected within a 1 minute(s) time period.

Analysis...
SEDANA aborted with an XTCA 531 as a response to a RDMP input.

Analysis of XTCA 531 in SEDANA has detected severity as HIGH...
SEDANA specific error code:
F2SUBSTR:SUBSTRING LENGTH GREATER THAN STRING

Abort Details...
Date  Time      No   Pid Func Ibserr Ibdiag Ibnote Apl Service      Severity Program
09MAY 10:18:11   1  3821 RDMP    531  52656   2251 VDA              HIGH     SEDANA
09MAY 10:18:09   1  3821 RDMP    531  52656   2251 VDA              HIGH     SEDANA
09MAY 10:18:07   1  3821 RDMP    531  52656   2251 VDA              HIGH     SEDANA
09MAY 10:18:06   1  3821 RDMP    531  52656   2251 VDA              HIGH     SEDANA
09MAY 10:18:03   1  3821 RDMP    531  52656   2251 VDA              HIGH     SEDANA
```

Abort Details…
The Abort Details is a list of the actual aborts causing this alert. They are shown along with their most essential diagnostic information. To ensure that the analysis text does not grow too large, the list is limited to a maximum of 10 aborts. An **Alert Message Analysis** may also contain entries of aborts that are similar in characteristics.

Persisting Abort Details...
This is identical to the **Abort Details…** only the word Persisting is added to inform on the fact that these aborts are identical to aborts that have already been reported within the last hour and the aborts are persistently occurring. In this case the original aborts reported are shown using the below section name.

Original Aborts Reported (Alert 1)...
If an **Alert Message** was raised and the problem is still occurring, another Alert Message may be raised. To inform on the fact that this is an already known problem that has been reported, this section is used to inform of the original **Alert Message** and the aborts that caused the original alert.

Similar Aborts (Alert 1)...
The Alert Message Analysis may find aborts that are not completely identical, but are very similar in nature - with these aborts having also caused an Alert Message to be raised. These similar aborts are shown here. (See ***Classifying Aborts as Identical or Similar***)

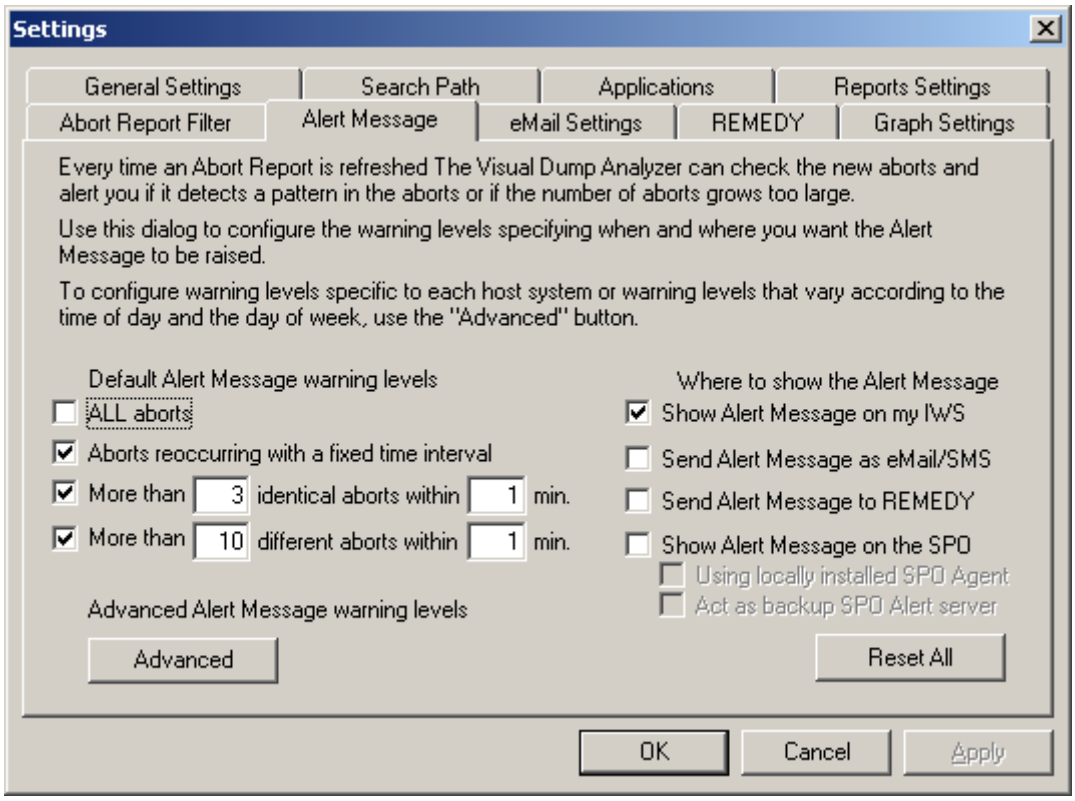
Suppressing Alert Messages
If an **Alert Message** has been raised and the aborts are still occurring, the new aborts may reach the same **Alert Message Warning Level** again very quickly. To avoid raising the same alert repeatedly, **Alert Messages** will be suppressed if the **Alert Message Analysis** detects that an identical alert has been raised in the past hour.

When the same **Alert Message Warning Level** is reached, it will cause a new **Alert Message Analysis** to be produced, but the **Alert Message** will not be raised. For these new aborts, The *Alert Id* column in the Abort Report will be updated with the Alert Id number, but the number will be shown in parenthesis – e.g., (3). This notation is used to inform that the **Alert Message Warning Level** was reached again but the **Alert Message** was suppressed.

Configuring Alert Message Warning Levels

Every time the Abort Report is refreshed, The Visual Dump Analyzer will check the new aborts and alert you if it detects a pattern in the aborts, or if the number of aborts grows too large. The **Alert Message Warning Levels** are used to customize the frequency of aborts and the pattern of the aborts that you want to cause an Alert Message.

To customize your warning levels, open the below dialog by selecting "**Tools | Settings | Alert Message**". Then adjust the individual levels and check the patterns you want to be active.



The four **Alert Message Warning Levels** defined by this dialog are the default values used for all systems that do not have an advanced setting. These default warning levels should be sufficient for most users, however, you can set more advanced levels for specific systems. The advanced sitting will allow you to configure the same four **Alert Message Warning Levels**, but with individual settings per system. Further, you can specify different values depending on time of day and day of week.

Warning on All Aborts

When selected, *ALL aborts* will cause an **Alert Message** to be raised whenever an abort occurs. The **Alert Message Analysis** may group more aborts into the same **Alert Message** if they are found during the same analysis. Because this setting covers all cases of aborts, it cannot be combined with the other warning levels.

Warning on too many identical aborts

Being the most commonly reached warning level, this level detects when a specific program errors multiple times in the same way. When this level is reached, it is usually an indication that a single specific problem within one application is occurring.

Warning on too many different aborts

When this level is reached, it is usually an indication of a more serious problem that is not limited to a specific program. Instead, it often affects more programs that may not even belong to the same application. In general, it is likely to be an error in shared functionality or shared database contents.

If this warning level is reached and the aborting programs span three or more applications – of which SYS is one – the Alert Message Analysis will deem all these aborts to relate to a SYS problem and request SYS staff to be alerted rather than staff from the aborting applications. In case SYS is not part of the aborting applications but the number of aborts have exceeded this warning level by a factor of 3, this problem will also be deemed to be a SYS rather than an application problem.

Warning on reoccurring aborts

This warning level will detect aborts originating from queues or Time Call. If an abort occurs 3 or more times and the time intervals between the aborts are the same, this warning level is considered to be reached. When this occurs, it is usually an indication of a minor problem limited to a specific queue or program. However, this problem is not expected to disappear without human intervention as it has already happened at least 3 times and it is expected to keep occurring with the same time interval until action is taken.

When this warning level is not set, this type of error is normally easily overlooked as is can be difficult to detect the problem when it is not noticed that it a specific abort is reoccurring every hour or daily.

These latter three warning levels can and should be combined. However, once a specific abort has been included in one **Alert Message,** it is automatically prevented from occurring in another **Alert Message**. This prevents the same abort from causing more than one **Alert Message** to be raised.

Note that if a **Report Filter** is applied, only aborts allowed in the Abort Report by the filter will be checked and counted by the **Alert Message Analysis** and compared with the warning levels.

Even if an abort is allowed in the **Abort Report** by the filter, it will not be checked or counted by the **Alert Message Analysis** and compared against these warning levels if it is marked as not being able to raise **Alert Messages** in the **VDA Knowledge Database**. (See ***Showing Alert Restrictions*** earlier in this document).

Running in Monitoring Mode

The Visual Dump Analyzer can be used to monitor various systems with abnormal conditions generating Alert Messages to the interested parties. These Alert Messages can be raised in four different ways:

1) As a small Pop-up window on your local Workstations (IWS)
2) As eMail or an SMS to a predefined mailbox or mobile phone.
3) To the REMEDY Action Request System making it a REMEDY Ticket.
4) On the Unisys Operator SPO (Single Point of Operation).

When The Visual Dump Analyzer is running in Monitor Mode, it will prepare itself for continuous execution. This includes ensuring that the raw data gathered from the mainframe to produce Abort Reports are not kept in memory once they are used, thereby ensuring that the IWS does not drain itself of memory. As a consequence of this, you cannot request The Visual Dump Analyzer to save the Abort Report to a local disc while it is running in Monitor Mode as it will have already freed the raw data for the report.

To place The Visual Dump Analyzer in Monitor Mode, select "**Tools | Settings…**" from the main menu, then select the "**Report Setting**" pane and finally select "**Enable continued execution (Monitor Mode)**".

Even though The Visual Dump Analyzer can run continuously when placed in Monitor Mode, it is recommended that the Workstation running in Monitor Mode be booted about once every week. This helps ensure that Windows or other applications running on the Workstation are not causing problems and thus preventing The Visual Dump Analyzer from properly monitoring the USAS applications.

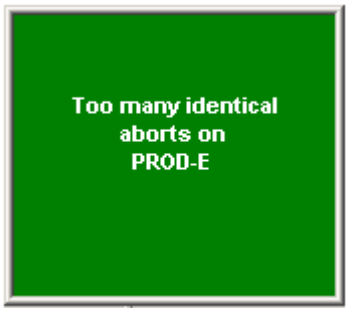
The desired method for being informed of an **Alert Message** can be configured under "**Tools | Settings…**" within the main menu.

Raising Alert Messages on your local Workstation (IWS)

This method raises the Alert Messages on your local Workstation and is meant for personal use. Being the most common method used for raising alerts, it allows you to set the warning levels in any way that you wish as they will only apply to you. Moreover, the levels that you choose for the development, test and production systems will not affect other instances of The Visual Dump Analyzer that are running in Monitor Mode and raising Alerts to the SPO or REMEDY for production systems.

With this method, you can produce a **Report Filter** that will limit the monitoring and thereby the **Alert Messages** that are raised to your IWS. This filter can monitor a limited number of programs and simply ignore the rest. For example, if you've just updated a program and have placed it on a test system, you can have The Visual Dump Analyzer monitor this specific program and inform you if it starts to abort, thus allowing you to be the first that is aware of this condition.

An **Alert Message** is raised by animating a small warning window in the bottom right corner of your screen. This window will be shown as the top-most window so that it will always be visible. It will not, however, take the input focus so you can continue working without interruption.



You can configure the appearance of this small warning window in the "**Report Settings**" pane and even test its appearance by pressing **Show Test Alert**".

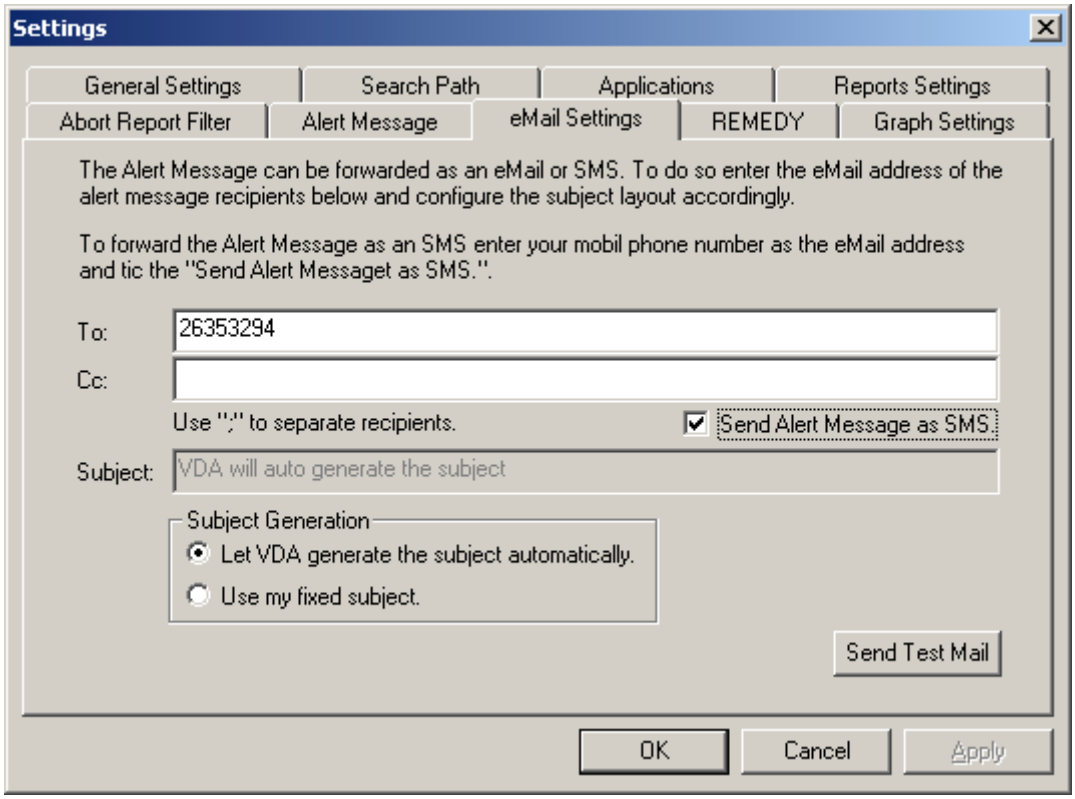
The **Alert Message** will be raised even if The Visual Dump Analyzer application is in the background or is minimized at the time.

Once the **Alert Message** is raised, you can use it to quickly activate The Visual Dump Analyzer application by clicking the warning text itself. The Visual Dump Analyzer will be maximized and brought to the foreground along with maximizing the Abort Report that has raised the **Alert Message**. If you instead click the non-text area within the warning window, it will simply close without activating The Visual Dump Analyzer.

It is not required that The Visual Dump Analyzer is running in Monitor Mode to raise **Alert Messages** on your local Workstation.

Raising Alert Messages as eMail or SMS

You can configure The Visual Dump Analyzer to send the Alert Message as eMails or SMS message to promptly inform you when an Alert has been raised.



When you want to send the Alert Message as an SMS, just enter the mobile phone number in the **To:** field and select "**Send Alert Message as SMS**" as shown above.

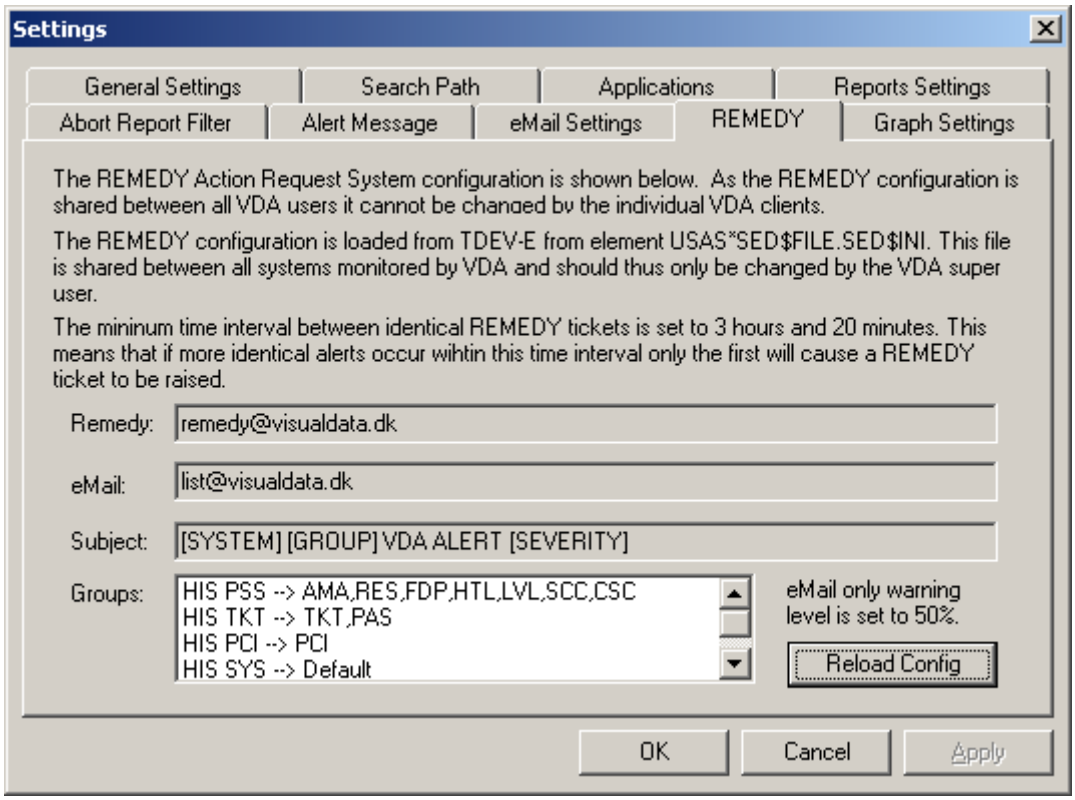
To enable The Visual Dump Analyzer to raise Alert Messages via eMail or SMS, an email server must be configured in the [EMAIL] section of the general configuration element (USAS*SED$FILE.SED$INI) on the default mainframe server. If the [EMAIL] section is missing or it does not name an email server, the above "**eMail Settings**" pane will not be visible.

It is not required that The Visual Dump Analyzer is running in Monitor Mode to raise **Alert Messages** via eMail or SMS.

Raising Alert Messages as REMEDY Tickets

To raise Alert Messages via the REMEDY Action Request System, a REMEDY configuration must exist in the [REMEDY] section of the general configuration element (USAS*SED$FILE.SED$INI) on the default mainframe server. Because the REMEDY configuration is shared among all users, you cannot change the actual REMEDY configuration using this dialog. This dialog only displays the present REMEDY configuration.

To configure the REMEDY information shown below, you must contact the super user for The Visual Dump Analyzer and have him change the REMEDY configuration in the general configuration element on the default mainframe server.



To enable The Visual Dump Analyzer to raise Alert Messages via REMEDY both an email server [EMAIL] and a REMEDY configuration [REMEDY] must exist in the general configuration element. If either section is missing or it does not name an email server, the above "**REMEDY**" pane will not be visible.

Even though it is not required that The Visual Dump Analyzer is running in Monitor Mode to raise **Alert Messages** to REMEDY, you should not configure your local workstation to raise REMEDY Tickets without prior approval from the person responsible for your REMEDY system.

Raising Alert Messages on the SPO

The Visual Dump Analyzer can raise the Alert Messages as alarms on the Unisys Operator Single Point of Operation (SPO) console. Alert Messages raised by The Visual Dump Analyzer appear on the SPO just like other alarms that are raised and require action by the Unisys Operator. When The Visual Dump Analyzer has raised an alarm on the SPO and the Unisys Operator opens the alarm, he will be shown the entire Alert Text informing him about the problem along with details of the person they should contact.

The Visual Dump Analyzer accesses the SPO via the VDA SPO Agent (VdaSpo.exe). This is a Windows agent that provides the interface between all instances of The Visual Dump Analyzer that wish to raise Alert Messages (or other messages) on the SPO and the SPO software provided by Unisys. The VDA SPO Agent resides on the SPO Windows server and accepts incoming DCOM calls from the Visual Dump Analyzer GUIs located on various Workstations and servers. It then pushes these messages to the SPO using the SPO Windows API.

The VDA SPO Agent will collect and keep a local copy of all the messages raised on the SPO. This enables the VDA SPO Agent to provide a history of all messages raised on the SPO no matter what instance of The Visual Dump Analyzer raised the individual messages. This history of messages can be fed back to any instance of The Visual Dump Analyzer and displayed in report form allowing them to re-open and view all historical messages raised on the SPO. (See ***Show SPO Alerts***).

The Visual Dump Analyzer must be running in Monitor Mode in order to raise **Alert Messages** on the SPO,

Running VDA in backup mode

When The Visual Dump Analyzer is configured to raise Alert Messages on the SPO, you can configure a second instance of The Visual Dump Analyzer as a backup server and have it raise the Alert Messages on the SPO if for some reason the primary server should fail. The backup server can be located anywhere including a disaster/recovery center so it is ready to automatically take over in case of problems.

In order to initiate the disaster/recovery configuration where two instances of The Visual Dump Analyzer are monitoring the same systems and raising Alerts Messages on the SPO, both instances of The Visual Dump Analyzer must be running on predefined servers. The two servers are configured in the [SPO] section of the general configuration element (USAS*SED$FILE.SED$INI) on the default mainframe server.

The [SPO] section defines two VDA SPO servers named **VdaSpoServer** and **VdaSpoServer2**, which inform The Visual Dump Analyzer of the primary and backup servers, respectively. Further, the backup instance of The Visual Dump Analyzer running on the server specified by **VdaSpoServer2** must have "**Act as backup SPO Alert server**" set. This option is found in the "**Alert Message**" pane in the "**Settings**" dialog opened by "**Tools | Settings...**" from the main menu.

Once the two servers are configured and placed in Monitor Mode, they will both monitor the USAS Systems from which **Abort Reports** have been opened. Each time either server accesses a monitored system, it will place a time stamp in the VDA common bank (SED$Q) on that system to indicate that the operation was successful. With a unique time stamp being written by the primary and backup servers, they can each verify that the other server is running and has completed its monitoring processes by checking these time stamps.

If the backup server detects that the primary server has not updated its time stamp for more than 15 minutes, it will take over the responsibility of raising **Alert Messages** on the SPO. Further, it will raise an **Alert Message** to the SPO informing the operator that the backup server has taken over as the primary server and that the primary server appears to not be running. When the backup server detects that the primary server is again up and running successfully, it will switch over to backup mode and once again leave the responsibility of raising the **Alert Messages** to the primary server. Also, it will raise a new **Alert Message** to the operator informing him that all is now back to normal and that the primary server is running again.

Similarly, if the primary server detects the backup server time stamp to be more than 15 minutes old, it will naturally continue raising **Alert Messages**, but it will raise another **Alert Message** informing the operator that the backup server has failed. Once the primary server detects the backup server is again up and running, it too will raise an **Alert Message** informing the operator that all is now back to normal and the backup server is running again.

Two servers can be monitoring as many systems as you wish, but it is imperative that they are monitoring the exact same systems. Otherwise, one server will conclude that the other one has failed because its time stamps are not present for the system that is only being monitored by an individual server.

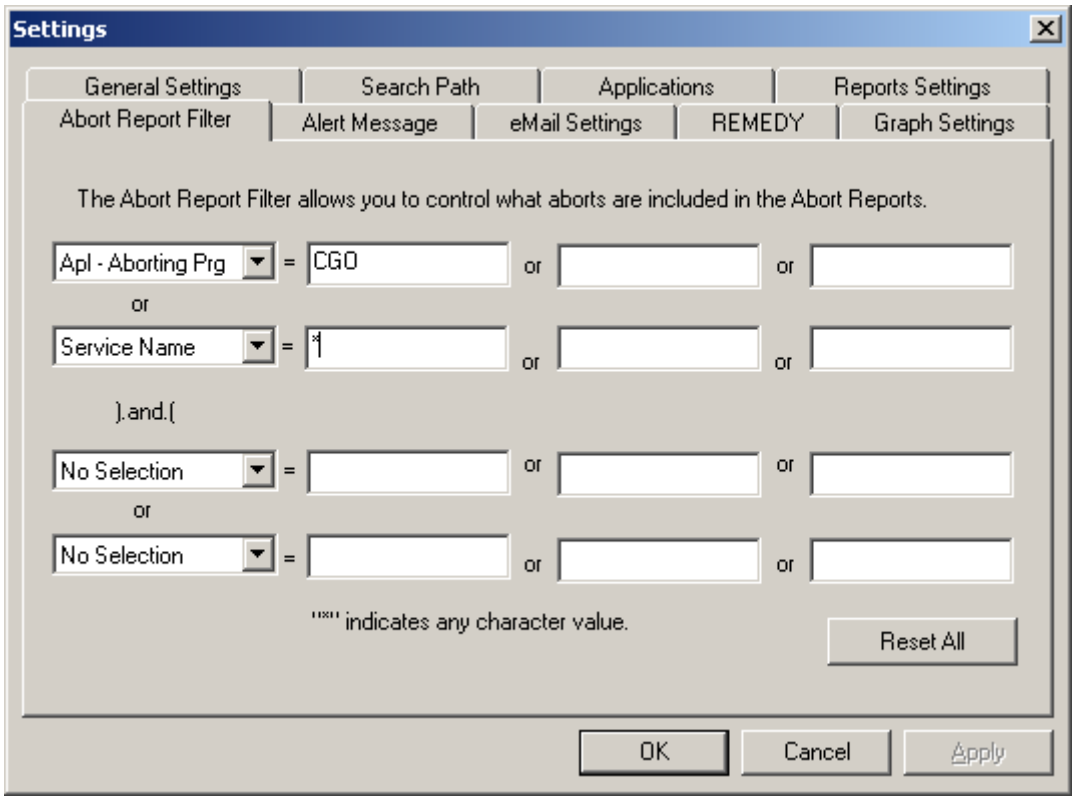
This method of handling disaster/recovery ensures a simple and safe way of handling cases when one server is not running or cannot access a mainframe for any reason. These procedures also ensure that the situations are handled without manual intervention – both in the disaster situation and when recovering back to normal operation mode again.

Setting Reports Filters

When Abort Reports are retrieved from the mainframe, they will always contain information on all aborts taken during the time period covered by the Abort Report. To limit the aborts actually allowed in the Abort Report, you can apply a **Report Filter** to the Abort Report.

Once the **Report Filter** is defined and applied, only aborts that meet the criteria defined in the filter will be shown in the Abort Report.

To define a **Report Filter,** open the below dialog by selecting "**Tools | Settings | Abort Report Filter**".



Once the **Report Filter** is defined, you can activate or deactivate it by using the Pop-up menu in the Abort Reports. Further, you can use the **Report Filter** to permanently limit the Abort Reports to a specific application or simply to search the Abort Reports for a specific program.

Command line options

You can use the command line to make The Visual Dump Analyzer automatically open one or more Abort Reports or Files Reports when it is initially opened.

In the Windows shortcut that is used to open The Visual Dump Analyzer (Vda.exe), add the following:

…/Vda.exe [**/R<no of days>:<system-id>] [/M<no of days>:<system-id>] [/F<apl>:<system-id>]**

The **/R** and the **/M** option both use the same data sequence. The first data item is a numeric value defining the number of days that is covered by the Abort Report and the second is the system name from which to read the Abort Report.

Multiple Abort Reports can be initially open by simply adding an entry in the command line for each report. Abort Reports that are opened using the **/R** option will all be shown as tiled windows, whereas Abort Reports opened using the **/M** option will be opened with the window being minimized.

To open a File Report, use the **/F** option followed by the application name (e.g., ALL, RES, TKT) and system-id. Finally, File Reports and Abort Reports can be combined in any manner.

Saving Abort Reports to disc

Once an Abort Report has been read from the mainframe, you can save it to a local disc so that it can be retrieved later. For archival purposes, it is a good idea to save Abort Reports on a daily or weekly basis. Because the SED$Q common bank will wrap around, periodically saving Abort Reports to a local disc is the only way to keep a complete history of aborts.

To save the Abort Report to disc, select "**File | Save Abort Report to Disc**", or click the "**Diskette**" icon and the normal Windows dialog for saving files will be displayed.

Opening Abort Reports saved on disc

You can retrieve an Abort Report that is saved locally by selecting "**File | Open Saved Dump or Report**" or clicking the "**Open Folder**" icon. This will display the normal Windows dialog for opening files. The Visual Dump Analyzer supports a multi-document environment, thus you can have multiple dumps or Abort Reports opened at the same time. Use the **Window** command in the main menu to arrange and navigate between open dumps.

Exporting Abort Reports

Abort Reports can be exported to CSV (Comma Delimited) files. By selecting "**Tools | Export Report to file**", you can save the report to the file name of your choice. Other Microsoft products such as Excel can then import this file for further processing.

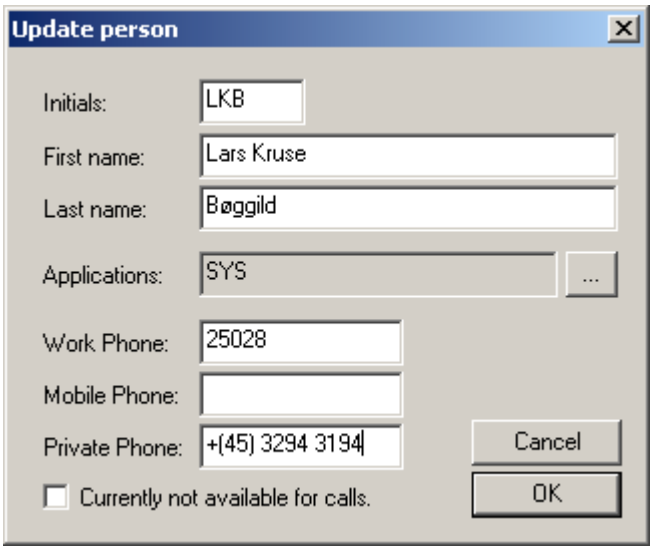
Emailing Abort Reports

Once saved to disc, Abort Reports can be attached to an email just like any other file. Alternatively, you can click the mail icon on the main menu and your configured mail client will be opened with the Abort Report being attached to a new e-mail that is prepared to be sent. This quick email option, however, requires that you have a Microsoft compliant email client installed.

The VDA Address Book

The Visual Dump Analyzer has an address book that identifies both personal and functional users within VDA. Being used everywhere VDA needs references to users, **The VDA Address Book** is stored on the host system and is shared between all VDA users. To view the Address Book select "**Tools | Address Book**" and the Address Book will be displayed. Once the Address Book is shown, it is can also be updated.

To update the Address Book, simply double-click a person in the Address Book. Further, by right-clicking in the Address Book window, you can select the "**Add New Person**", "**Update Person**" or "**Delete Person**" options. Selecting a person in the Address Book will open the dialog below.



Once the Address Book is updated, the changes are made both locally as well as distributed to all the Unisys host systems. Because the Address Book is shared between all users, it is important that the information within it is kept accurate.

It is recommended that you add yourself to the Address Book so VDA is able to identify you. To do this, first add yourself to the Address Book, then select "**Tools | Settings | Report Settings**" and set the "**User Identification**" to your name. If you don't identify yourself to VDA, you can still use all functions related to dump analysis, however, more Abort Report functions will be restricted as they are only allowed by identified users. E.g., Only identified user can change the **Abort Severity** from NORMAL. (See ***Abort Severity*** and ***Showing Alert Restrictions***).

Among other components, the Address Book is used to construct the Phone List in Alert Messages. If you do not want to be included in this Phone List, check the *Currently not available for calls* option, and you will be exempt from the generated Phone lists while still being identified within VDA. Moreover, you can use this option when you don't want to be notified by the Unisys Center Operators when Alert Messages are raised on the SPO.

The VDA Address Book is also used is in the "**Action by**" column in the Abort Report. This column provides the name - short name from the Address Book - of the person working the abort. Therefore, when you are viewing an abort, you can "mark" it as being handled by yourself – thus allowing others that are viewing the Abort Report to notice this and avoid spending time on the same problem.

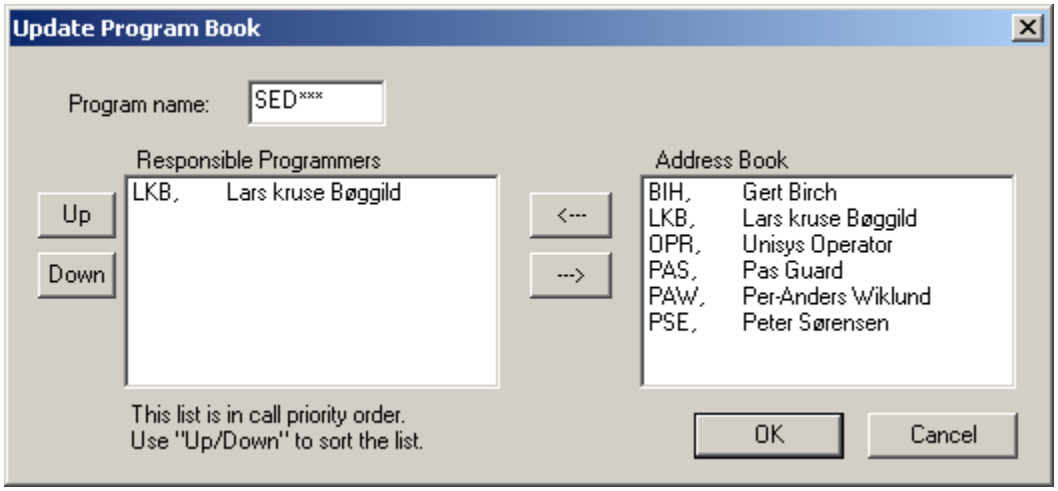
You can change the value of this column by double-clicking it and entering a dialog that allows you select a different person from the Address Book.

The VDA Program Book

The VDA Program Book is used to link responsibilities between programmers and specific program when such a relationship exists. This relationship is used by VDA to produce a more accurate Phone List in the Alert Message Analysis. That is, if a specific program causes an Alert Message to be raised on the operator console, the Alert Message will contain a Phone List that is firstly based on information from the Program Book and secondly from information from the Address Book.

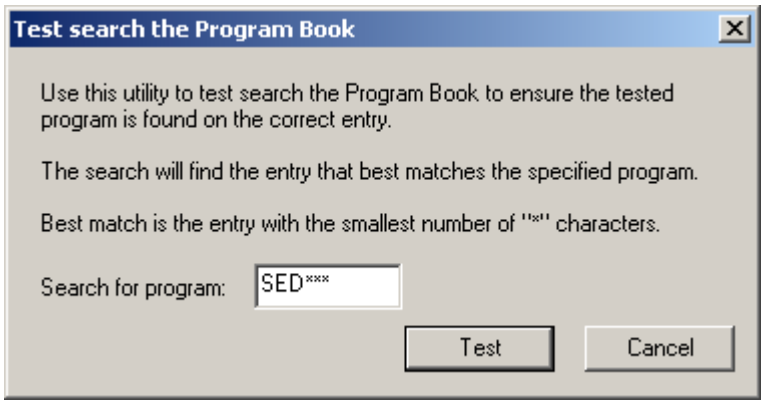
In short, if the aborting program exists in the Program Book, only the programmer that has specific responsibility for the program will be included in the Alert Message Phone list. If it does not exist, responsibility is assumed to be shared between all the programmers in the Address Book that are working in the application of the aborting program. (See ***Alert Message Analysis)***.

To view the Program Book, select "**Tools | Program Book**" in the main menu and the Program Book will be displayed. Just like the Address Book, the Program Book is shared between all VDA users. To update an entry in the Program Book, simply double-click it. Furthermore, you can right-click within the Program Book window and select the "**Add New Program**", "**Update Program**" or "**Delete Program**" features. This dialog below allows you to update the Program Book.



Note that program names in the Program Book can contain an asterisk ("*") to indicate a wild card.. When the Program Book is searched during Alert Message Analysis, only the entry that best matches the program name will be used when multiple entries match the aborting program. The best match is defined to be the entry with the least number of asterisk characters.

To ensure that your updates to the Program Book give the expected results, you can perform a test by searching the Program Book. To do this, right-click the Program Book window and select "**Test Program**".



By clicking "**Test**", the entry that best matches in the Program Book will be colored **GREEN**.

Monitoring Freespace Files

The Visual Dump Analyzer can provide you with a complete overview of each freespace file on the system. It accomplishes this by combining the FCSS (File Control Superstructure) internal control information along with information controlled by USAS SYS and information gathered by The Visual Dump Analyzer itself during its Freespace Scans.

First, you can use The Visual Dump Analyzer to give you a quick File Report that informs you of how full each file is for all the FCSS freespace files on your system. Next, if a particular file stands out by, for example, being nearly full, you can receive a more detailed analysis of it.

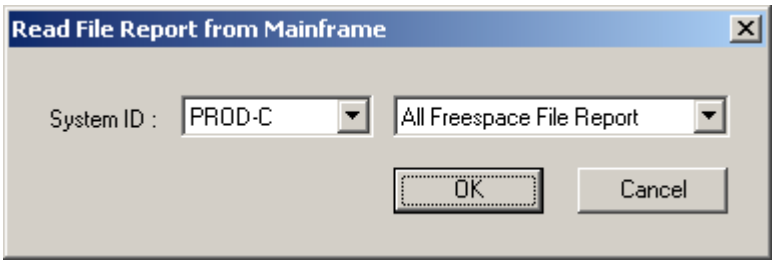
The File Report will be displayed as a bar graph that contains a bar for each FCSS freespace file in the selected application. The bars depict how much of each file is used for normal record allocation and how much is blocked by a Release Only setting. This graph provides all the essential information you need to get a quick overview of how each FCSS freespace file is setup along with the remaining amount of space that is still available for allocation. Because this information is gathered and analyzed at request time, it is always complete and up-to-date for all the files in the graph.

Second, the File Report can be used to monitor all freespace files used by the USAS applications on a given system and automatically raise an Alert Message if it detects a file that is growing too full.

Third and final, the File Report graph enables the user to quickly produce Detailed File Analysis for a specific freespace file. This analysis is presented in a number of detailed graphs, which contain virtually all the data that can be imagined about a file. By simply double-clicking the bar for a specific file, the file will be analyzed and all the detailed graphs related to it will be displayed. More files can be analyzed in detail at the same time to allow for easy comparison.

Reading File Report from the mainframe

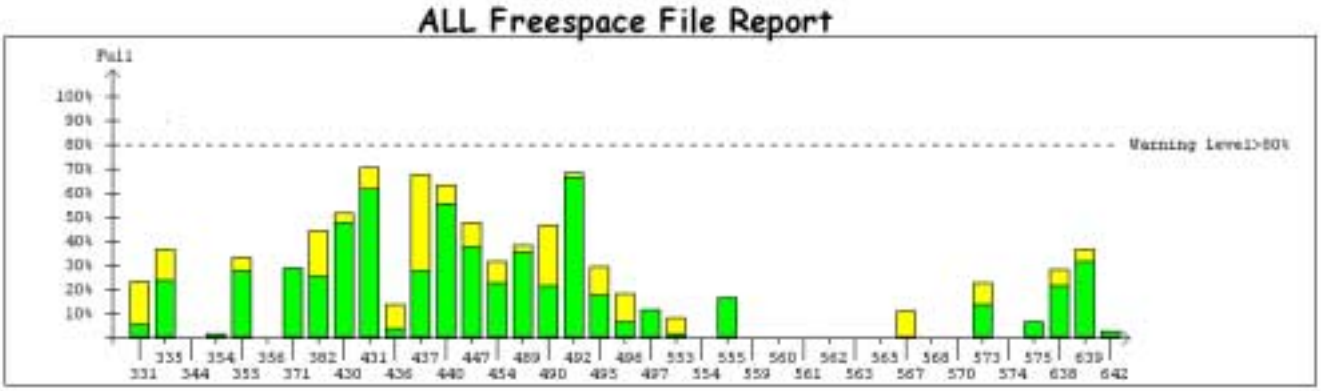
To read a File Report from the mainframe, select "**File | New File Report**" or click the "**File Report**" icon on the main menu. In the dialog, you can select the desired system along with the files to be included in the Files Report.



The File Report will retrieve information for the selected freespace files on the chosen system. The reports are essentially identical except for the actual files that they describe. The "**All Freespace File Report**" will contain ALL the freespace files on the system whereas a Freespace File Report for a particular application will only contain files within that application. An application report can be generated for each one that is known to The Visual Dump Analyzer.

File Reports

The File Report is initially shown as a Graph Report with all the files being presented by an individual bar. Moreover, you can toggle this into a more traditional line based File Report that contains one line of information for each file. You can use the line based File Report to update the file name information stored by The Visual Dump Analyzer for each freespace file.



The Visual Dump Analyzer uses colors to inform of the status of the individual files in the File Report. The **GREEN** color indicates that the file contains enough allocatable records to satisfy the Warning Level. The **RED** color indicates that the Warning Level has been exceeded. When a file has not yet been analyzed, the file will be shown in **GRAY**.

The **YELLOW** color represents the actual free records in the portion of the file blocked by Release Only bitmaps. Note that the **YELLOW** color will show the actual free records blocked by release only and will not count the allocated records in the Release Only blocked bitmaps. This means that the **YELLOW** color shows the amount of available space that will be gained by increasing the Release Only bitmaps.

If the record allocation in the file is controlled by a D-value that splits the file into two logical files, both logical files must satisfy the Warning Level for the file to be shown in **GREEN**. For example, if a file has a D-value of 7,75,8 and the Warning Level is at 80%, the file will surpass the Warning Level when one of the logical files have used more that 80% of the space reserved for that part of the file. That is, when 80% of the 75% portion below the D-value has been used or when 80% of the 25% portion above the D-value has been used, the file will have reached its Warning Level.

The files are identified by either their TIP file number or their USAS file number. To control the way the files are shown, right-click and select either "**Show TIP File numbers**" or "**Show USAS File numbers**". You can control the initial format of this display within the Graphs Setting pane in "**Tools | Settings | Graph Settings**".

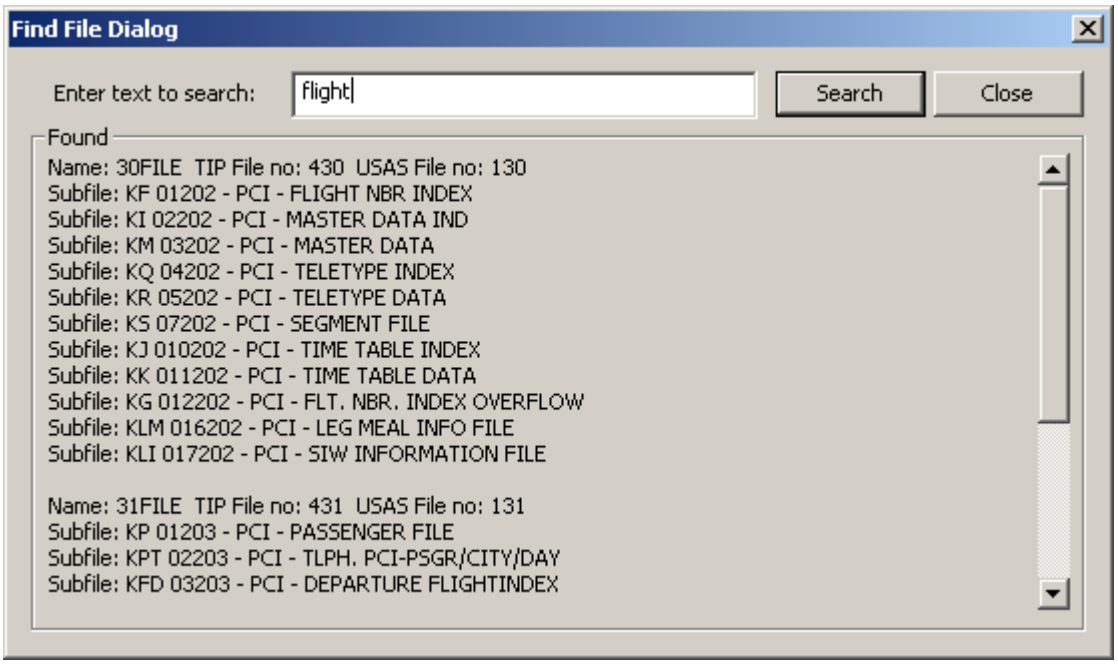
All the graphs shown in Files Report and in the Detailed File Analysis are normally shown using percentages instead of their actual values. This can be toggled by simply double-clicking the white space in the graph.

To obtain more detailed information for a given file, just single-click the bar representing the file you're interested in and actual file sizes along with information on subfiles in the physical file will be displayed.

```
Name: 30FILE  TIP File no: 430  USAS File no: 130  D-Value: 4,65,7  Tracks: 115.968
The file is 48,5% full, 955.484 of 1.855.488 basic records are available.
The file contains 65.257 (3%) free basic records marked as Release Only.
Subfile: KF 01202 - PCI - FLIGHT NBR INDEX
Subfile: KI 02202 - PCI - MASTER DATA IND
Subfile: KM 03202 - PCI - MASTER DATA
Subfile: KQ 04202 - PCI - TELETYPE INDEX
Subfile: KR 05202 - PCI - TELETYPE DATA
Subfile: KS 07202 - PCI - SEGMENT FILE
Subfile: KJ 010202 - PCI - TIME TABLE INDEX
Subfile: KK 011202 - PCI - TIME TABLE DATA
Subfile: KG 012202 - PCI - FLT. NBR. INDEX OVERFLOW
Subfile: KLM 016202 - PCI - LEG MEAL INFO FILE
Subfile: KLI 017202 - PCI - SIW INFORMATION FILE
```

Finding Files

When a file number for a specific file is not known or you want to find a file based on the data it contains, The Visual Dump Analyzer allows you to search among all files known to The Visual Dump Analyzer by entering free format text.

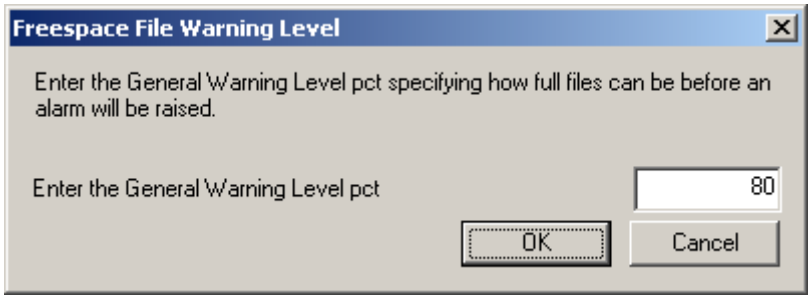


To open the File Find dialog, right-click to open the Pop-up menu and select "**Find File…**". This will open the above dialog where you can search for files. The search will be case insensitive and will include all files known to The Visual Dump Analyzer – even if the file is not included in the active File Report.

Setting Freespace Files Warning Levels

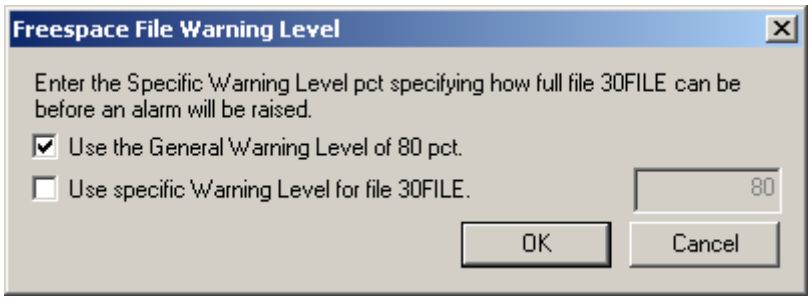
The Visual Dump Analyzer includes two types of configurable warning levels – a General Warning Level valid for all files on a given system and a specific Warning Level valid only for a specific file on a given system.

The General Warning Level is shown on the File Report as a dotted line across the graph. To set the General Warning Level, right-click the white space of the File Report and select "**Set Warning Level**" in the Pop-up menu. This will open the dialog below.



Note that the Warning Level set in this dialog is only valid for the system a File Report is being generated upon. This allows you to use different General Warning Levels on each system.

To set the Specific Warning Level for a specific file, right-click the bar related to the file you're interested in and select "**Set xxFILE Warning Level**". Alternatively, you can right-click the white space in one of the Detailed File Analysis graphs for the specified file. In either case, you will be directed to the dialog below.



In this dialog, you can control the specific Warning Level for the file. Again, note that the specific Warning Level is only valid for the specified file on the system the File Report is being generated upon.

If you set the specific Warning Level to zero, you will turn off all Monitoring for the file and thus no warnings will be raised for the file.

If a Warning Level is achieved and you have the "**Alert Me**" selected in the Pop-up menu, an Alert Message will be raised informing you that a file is becoming full. (See ***Raising Alert Messages on Freespace Files***)

Auto-refreshing File Reports

To ensure that the File Report is kept up to date, The Visual Dump Analyzer periodically checks the monitored system for changes in file statuses. In the Pop-up menu, you can toggle the refresh time period to be either 1, 2 or 5 hours – or you can turn the "**Auto Refresh**" logic completely OFF. Every time the File Report is refreshed, all the files are checked and if the Warning Level is found to be exceeded for one or more files, an Alert Message is raised. To manually refresh a File Report, press **F5** and the report will be immediately refreshed, even if "**Auto Refresh**" logic is turned OFF.

Raising Alerts Messages on Freespace Files

Raising **Alert Messages** is a very helpful feature that immediately informs you of when a file within the monitored applications starts becoming full. Whenever the refresh logic retrieves the latest file status, the report is analyzed and checked to verify if the **Warning Levels** have been exceeded. If the "**Alert Me**" option in the Pop-up menu is set and the analysis of the report reveals that one or more of the warning levels have been exceeded, an **Alert Message** is raised.

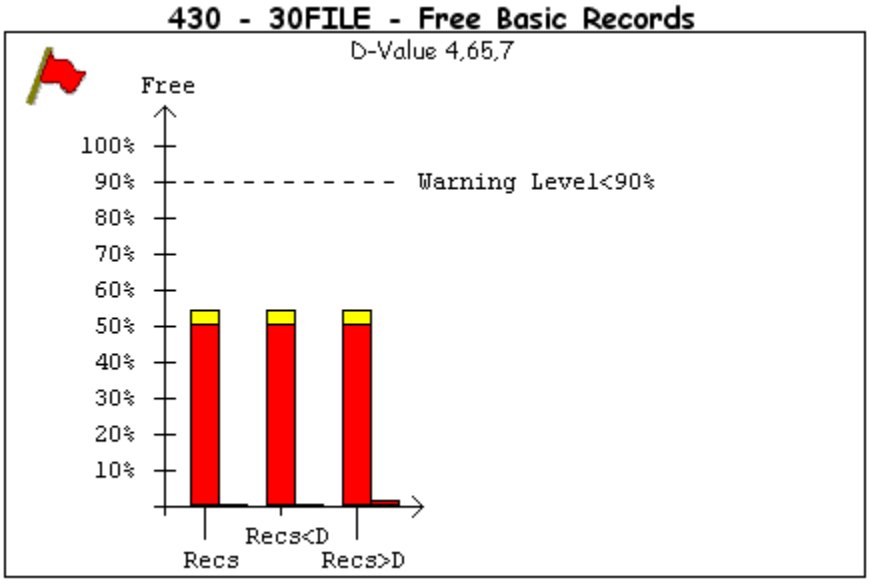
An **Alert Message** is raised by animating a small warning window in the bottom right corner of your screen. This window will be shown as the top-most window so it will always stay visible. It will, however, not take the input focus, so you can continue working without interruption.

The **Alert Message** will be raised even if The Visual Dump Analyzer application is not the active application at the time. If The Visual Dump Analyzer application is in the background or is minimized, the **Alert Message** will still be raised.

Once the **Alert Message** is raised, you can use it to quickly activate The Visual Dump Analyzer application by clicking the warning text itself. The Visual Dump Analyzer will be brought to the foreground and maximized with the File Report that has raised the **Alert Message** also being maximized to give you a helpful overview.

If the white (blank) space in the warning window is clicked instead of the text itself, the warning window will be closed and The Visual Dump Analyzer will not be activated.

When the "**Show Full Files**" is selected in the Pop-up menu, the Detailed File Analysis is automatically performed on the full file and a warning flag is shown in the top left corner of each detailed graph.



Graph-1

To view the **Alert Message Analysis** text describing the results of the analysis and explanation of the error, click the red warning flag.

Because the **Alert Message** logic is based upon an analysis of the new file status retrieved by the *refresh logic*, this feature must be turned ON. That is, turning the refresh logic OFF will cause the **Alert Message** to be turned OFF as well.

Raising **Alert Messages** for freespace files works just like raising **Alert Messages** for aborts. The configuration parameters in "**Tools | Setting | Alert Message**" that define how **Alert Messages** are raised apply to both aborts and files. (*See **Running in Monitor Mode***)

Alert Message Analysis for Freespace Files

The **Alert Message Analysis** provides the results of the Detailed File Analysis Report. This summary gathers information from the files themselves and the Severity Setting in the Graph Settings in "**Tools | Settings | Graph Setting**".

All this information is processed and the final result is used to generate a natural language text explaining the reason for the warning. To see this analysis, click the red warning flag in the Detailed Analysis Report graphs and the analysis will be shown in a yellow Pop-up window. You can close the **Alert Message Analysis** window by simply moving the mouse outside its border.

Besides a short header line identifying the Alert Message and showing its status, the analysis text is divided into the sections below:

Action…
This section informs you or the system operator of what action should be taken due to this **Alert Message**. This section can also inform you of who should be called to inform of the file situation. This information is intended for **Alert Messages** that are received by computer center operators rather than programmers.

```
VDA Application FreeSpace File Alert 1 from PROD-C (Raised at 13:31 LT.)

Action...
Please Call the File Group now.

FILES Phone list in random order...
VU-P, Vu-p Guard (Office hours 8-16) - Ext:4 7776 Mobile: Home:
LKB, Lars Kruse Baggild - Ext:4 5631 Mobile:2635 3294 Home:3294 3194

Reason...
TIP file 430 (30FILE) is running full on PROD-C.

Analysis...
The file is 48,9% full, only 946.887 of 1.855.488 basic records are available. The file
must have at least 90% free basic records, but the free PCT has dropped to only 51,0%.
This has been estimated to a severity level of MINOR. Current file size is 115.968 tracks
with a D-Value of 4,65,7. However, the file contains 65.257 (3%) free basic records marked
as Release Only that can be upped.

SubFile Info..
Subfile: KF 01202 - PCI - FLIGHT NBR INDEX
Subfile: KI 02202 - PCI - MASTER DATA IND
Subfile: KM 03202 - PCI - MASTER DATA
Subfile: KQ 04202 - PCI - TELETYPE INDEX
Subfile: KR 05202 - PCI - TELETYPE DATA
Subfile: KS 07202 - PCI - SEGMENT FILE
Subfile: KJ 010202 - PCI - TIME TABLE INDEX
Subfile: KK 011202 - PCI - TIME TABLE DATA
Subfile: KG 012202 - PCI - FLT. NBR. INDEX OVERFLOW
Subfile: KLM 016202 - PCI - LEG MEAL INFO FILE
Subfile: KLI 017202 - PCI - SIW INFORMATION FILE
```

Phone List…
The Phone List displays the responsible application programmers along with their phone numbers. The responsible programmers are selected from the Address Book. Further, all people in the Address Book that are identified as being responsible for "**FILES**" are selected and made part of the Phone List.

Reason…
This section informs of what file has caused this **Alert Message**.

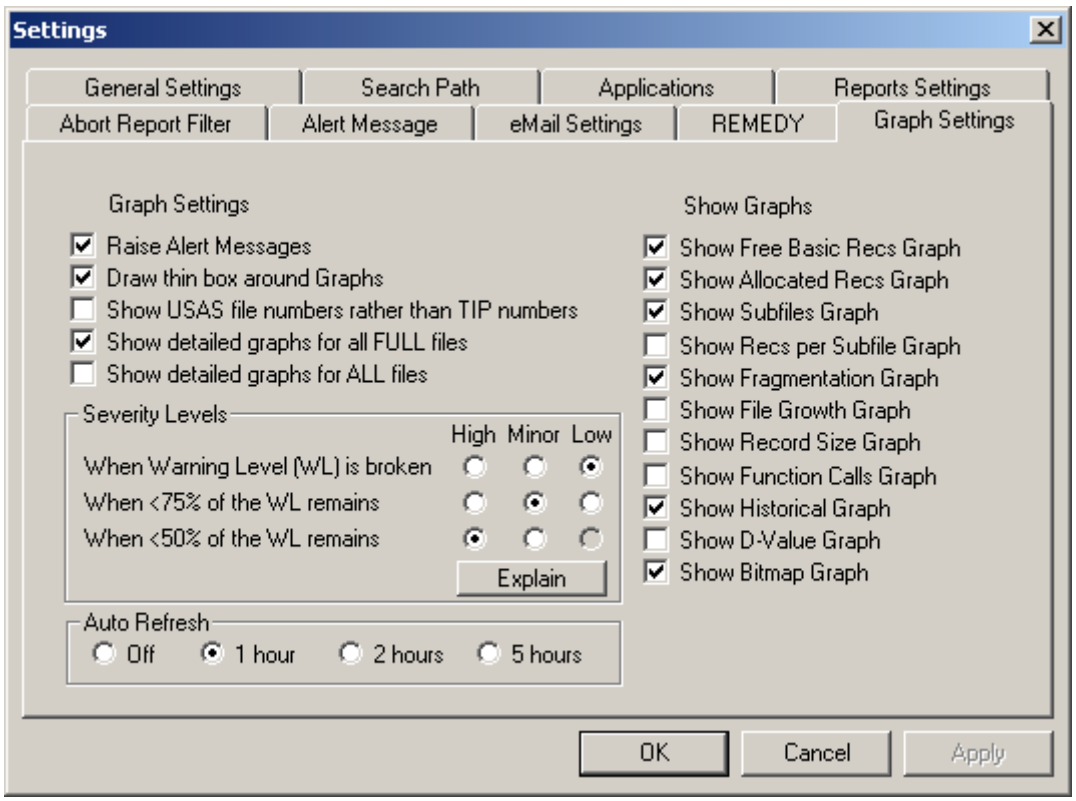
Analysis….
The analysis is natural language text describing the file situation in as much detail as possible. The text will depend on the file and the setting. (*See **Configuring Alert Message Warning Levels for Freespace Files.***)

Subfile Info….
This section describes the data that is within the file that is becoming full. Because TIP file numbers are usually not as helpful when analyzing this type of problem, the actual subfiles and their data content are listed in order to help determine the consequences that a full file may bring.

Once an **Alert Message** for a freespace file has been raised, a new **Alert Message** will not be raised for the same file within the next 24 hours unless the new **Alert Message** has a higher severity.

Configuring Alert Message Warning Levels for Freespace Files

To configure how File Reports are initially opened in The Visual Dump Analyzer, use the Graph Setting pane found in "**Tools | Settings | Graph Setting**". This will open the below dialog allowing you to control the way File Reports are shown along with how **Alert Messages** are raised for freespace files.



Warning Levels for Freespace Files
The main Warning Level is either set as a general Warning Level valid for all files or as an individual Warning Level set for a specific file. This Warning Level indicates how full a file can be before an alarm will be raised. If, for example, the Warning Level is set to 80%, it indicates that when 80% of all basic records in the file are used (or blocked by Release Only bitmaps), an alarm will be raised. That is, in this case at least 20% of the basic records must be available for new allocations.

This 20% is called the "Minimum Available Warning Level" of the file. Two automated Warning Levels exist, which are calculated on the "Minimum Available Warning Level". The first warning is when less than 75% of this level is available and the next when less than 50% of it is allocatable. *(See **Alarm Method 1 - Early Warnings**)*

Minimum Available Warning Level.
Warning Level <75% of the "Minimum Available Warning Level" is allocatable.
Warning Level <50% of the "Minimum Available Warning Level" is allocatable.

Freespace File Severity Codes

Alarms raised on Freespace files can have three different severity codes with each code requesting action by the people responsible for files (i.e., the people marked with “FILES” in the address book). Further, alarms will always be raised when the Warning Level is surpassed. If this happens during normal working hours (Monday through Friday 8-16), the severity codes will all request immediate action. If the Warning Level is exceeded outside of normal working hours, however, the severity code will determine when the action will be taken.

HIGH - Request action now - around the clock.
MINOR - Postpone action outside working hours until next morning.
LOW - Postpone action outside working hours until next working day.

Alarm Method 1 – Early Warning

By setting the Warning Level to a level that is considered not too critical and the severity code for the Warning Level to LOW, an alarm of severity LOW will be raised early – although it will not request action outside normal working hours. In this way, the alarm functions like an early warning. Once an alarm has been raised, a new alarm for the same file will not be raised again during the next 24 hours unless the severity level becomes higher. By setting the <75% Warning Level to MINOR, a new alarm will then be raised (of severity MINOR) if the available space drops to less than 75% of the “Minimum Available Warning Level”. Similarly, setting the <50% Warning Level to HIGH will ensure that a new alarm of severity HIGH is raised when the available records drops to less the 50%.

As a result, setting the Warning Level to 80% with severity LOW, the <75% to MINOR and <50% to HIGH will result in the following. An early warning alarm (Severity LOW) will be raised once the file has less than 20% available records, a new alarm (Severity MINOR) will be raised when the file has less than 15% available records and again (Severity HIGH) when the file has less than 10% available records.

Alarm Method 2 – Simple Alarms

By setting the Warning Level to a level that is considered critical and the severity code to HIGH, an alarm of severity HIGH will be raised immediately when the Warning Level is exceeded. This is a simple and effective way to raise alarms as no further alarms will be raised on the file during the next 24 hours regardless of the settings for <75% or <50%. This is because an alarm with the highest possible severity has already be raised.

Detailed File Analysis

Once a File Report has been retrieved from the mainframe, it can be used to request a Detailed File Analysis performed on any of the files in the File Report. The Detailed File Analysis will analyze the file and develop a number of different detailed graphs with each showing a specific aspect of the file. You can control how many of these graphs are initially shown by selecting them in the Pop-up menu.

To request a Detailed File Analysis to be produced for a given file, simply double-click the bar representing the file in the File Report. You can have as many files shown in the Detailed File Analysis as you choose by continuing to double-click more files. If more than one file is included in the Detailed File Analysis, all of the individual graphs for each file are shown next to each other for easy comparison. When a single file is only selected, the detailed graphs are shown over multiple lines to allow for as many graphs to be visible at a time.

In the Pop-up menu, you can select either "**Show Full Files**" or "**Show All Files**". This will cause either all full files or simply all files to be automatically included in the Detailed File Analysis.

When a file is included in the Detailed File Analysis, it can be excluded by use of the Pop-up menu.

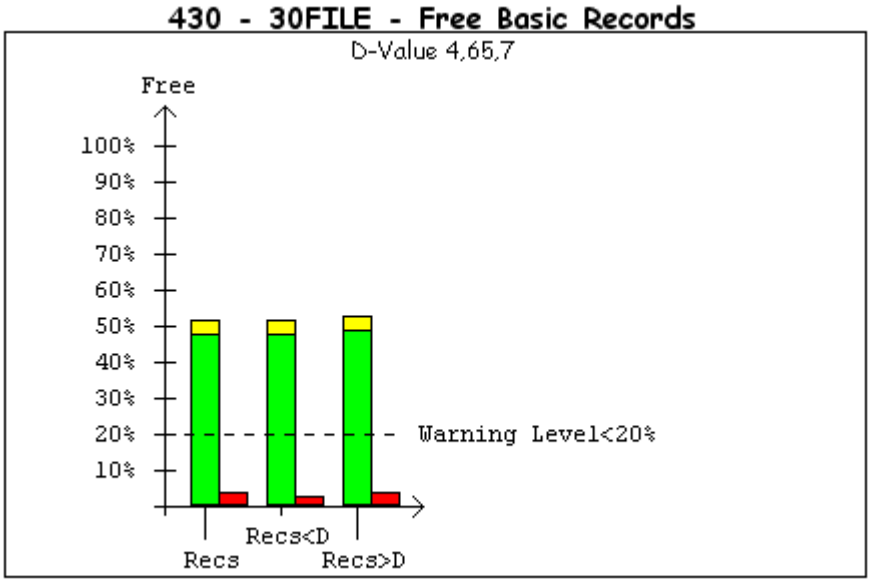
The Detailed File Analysis contains a number of bar graphs with each informing about different aspects of the specified file. To obtain detailed information on the value depicted by a bar, click on it and a yellow box window will display the specific values of the bar along with their meaning.

Single-clicking the white space of a detailed graph will display a yellow box window informing of the basic information of the file along with subfile information.

Double-clicking the white space of a detailed graph will cause the graph to toggle between percentage values and actual values.

Detailed File Analysis – Graph 1

The first graph informs of the amount of Free Basic Records (**Recs**) in the file. When an FCSS freespace file contains a D-value that is not 100, the file is actually divided into two logical files – the first being below the D-value and the second being above the D-value. In this case, The Visual Dump Analyzer will analyze each logical part of the file as if they were separate files.



Graph-1

As a consequence, the free **Recs** are shown using three sets of two bars.

The first bar depicts the counts for the entire file and is labeled **Recs.** It is colored GREEN because the Warning Level has not been exceeded, whereas if it had been the graph would be shown in RED. The portions of the file below and above the D-value are labeled **Recs<D** and **Recs>D**, respectively. It is important to realize that the Freespace Handler must select new records for allocation based upon the D-value setting, which may result in a record not being able to be allocated even though space is available on the other side of the D-value setting.

Further, note that Release Only bitmaps are determined and their free **Recs** are shown in YELLOW.

The second bar – shown here in RED – displays the net growth of basic records in the file since the last time the freespace handler statistic was reset. When the growth indicates a net allocation, the bar is shown in RED whereas a net de-allocation is displayed as GREEN.

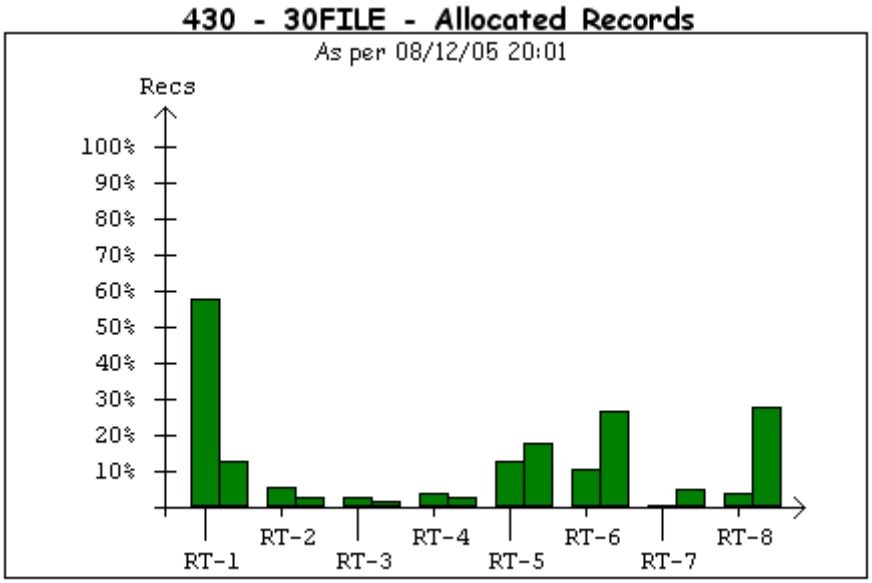
This graph gives a quick overview of the file usage while also showing if the D-value is set correctly. When the bars in this graph are almost identical in size, it is a good indication that the D-value is set correctly as the percentage of free basic records is almost the same both below and above the D-value.

If any of the bars in this graph get below the Warning Level, an **Alert Message** will be raised.

If you prefer to have the actual counts with this graph, just click the individual bars.

Detailed File Analysis – Graph 2

The second graph in the Detailed File Analysis shows the number of allocated records per record type. Each record type has two bars depicting the actual number of allocated records. The first bar shows the number of Logical Records within this record type and the second bar shows the number of Basic Records within this record type.



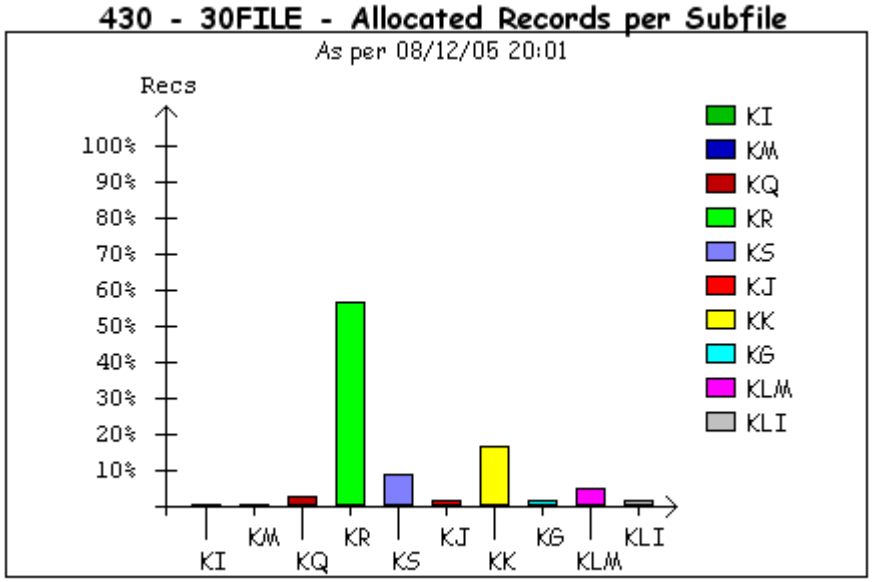
Graph-2

In this example, about 60% of the logical records in this file are used for record type 1 (RT-1) whereas this is equivalent to about 13% of all the Basic Records currently allocated in the entire file.

In order to display the actual counts for each record type, just click the individual bars.

Detailed File Analysis – Graph 3

The third graph displays the number of allocated logical records per subfile.



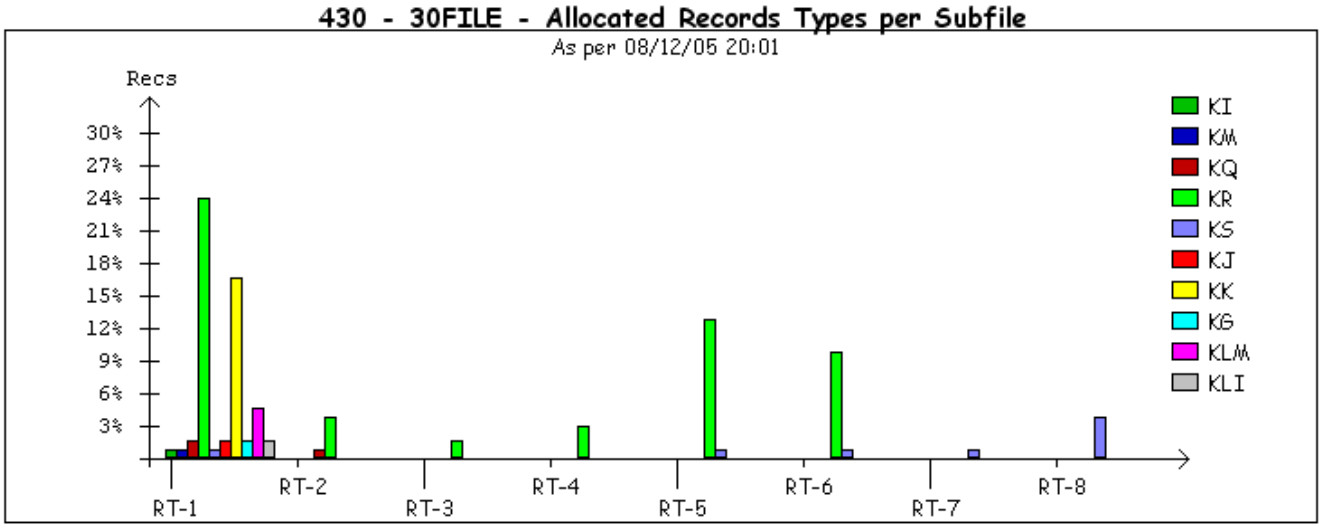
Graph-3

This graph along with the fourth graph gives a good overview of all of the record types that are allocated for the individual subfiles.

To view the actual counts for the subfile, click the individual bars.

Detailed File Analysis – Graph 4

The fourth graph shows how many records are allocated for each record type within each of the individual subfiles.



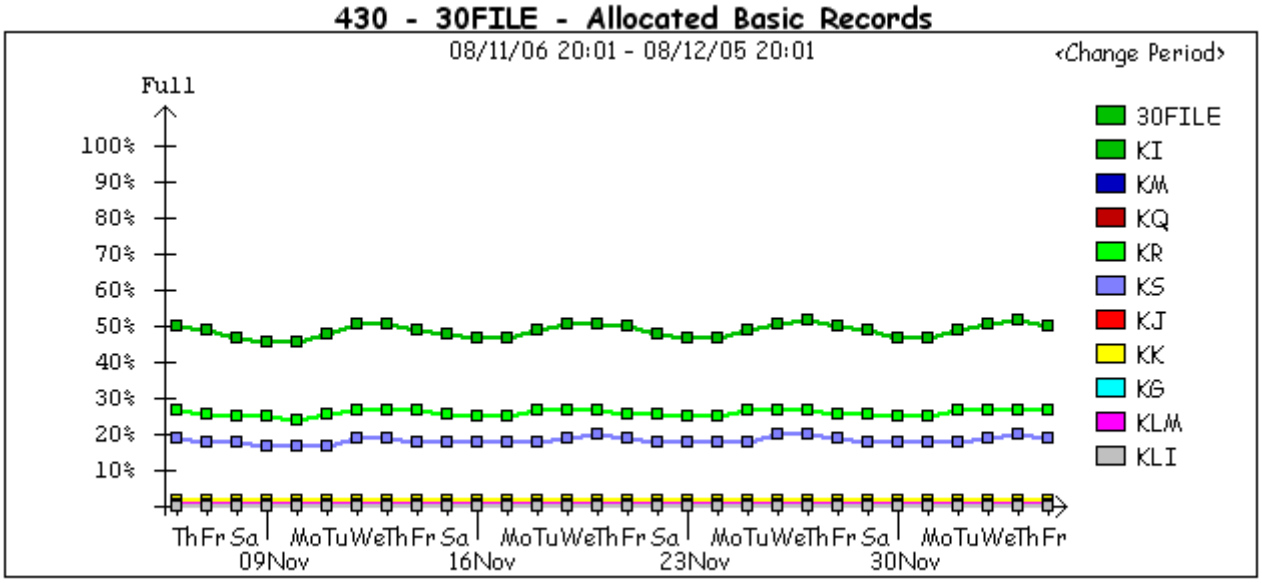
Graph-4

As with all of the graphs, double-clicking the white space within the graph will toggle the percentage values to actual record counts.

To view the actual counts for the record type and subfile, click the individual bars.

Detailed File Analysis – Graph 5

The fifth graph shows the historical development of the individual subfiles by displaying how the record usage has developed over time. In general, this graph conveys the cycles a subfile experiences along with how it is maintaining over time.

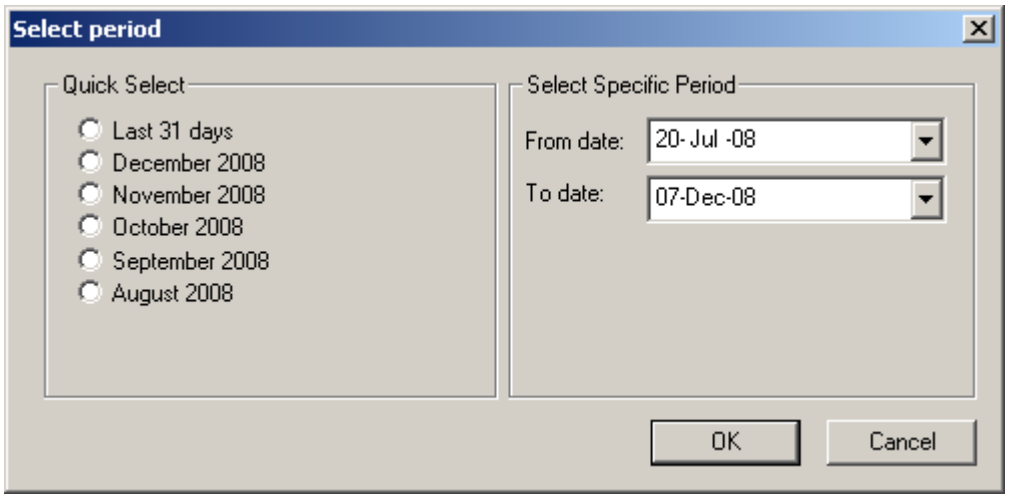


Graph-5

If a subfile is continuously growing over time, it is a strong indication that orphan records are being produced by the application. Likewise, when new development is introduced, if a new and unexpected pattern is observed with the file, it may indicate that application problems exist.

To view the actual count for a given day within a specific subfile, click the line squares.

By default, this graph will show the last 31 days. To change the period, click**<Change Period>** at the top left corner to open the dialog below.



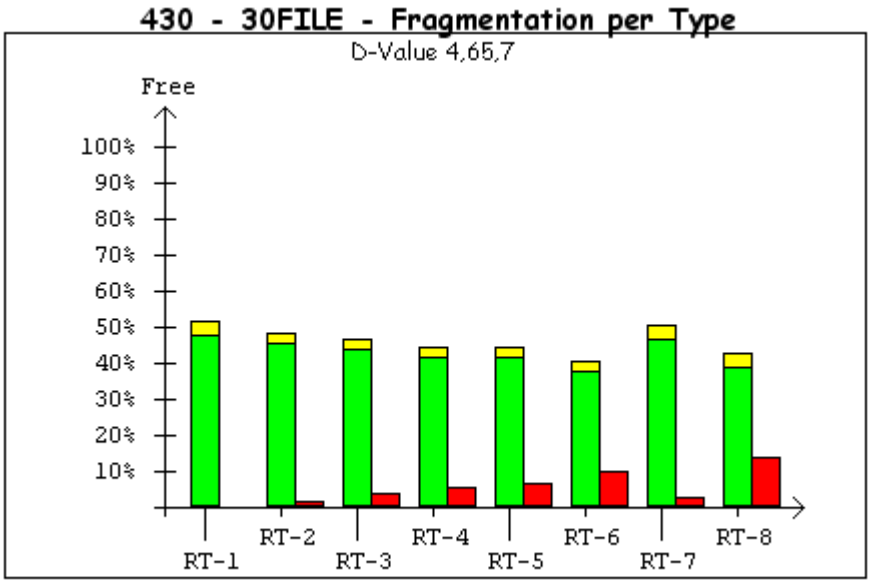
Use this dialog to change the period using Quick Select on the left or specify exactly what period is desired using Select Specific Period on the right.

This information is gathered by The Visual Dump Analyzer Freespace Scan process called FSSCAN. This process runs through all files daily and records this information so it can be presented in this graph. (*See **Freespace Scan***)

Detailed File Analysis – Graph 6

The sixth graph informs of the record fragmentation per record type. Due to the nature of FCSS files, all files having more than one record type are likely to get fragmented to some extent. However, some files may become much more fragmented as it can be difficult to estimate how many records are actually fragmented and how many are truly free and allocatable.

This graph shows two bars for each record type with the first bar showing the number of free records in GREEN and the number of Release Only records in YELLOW.



Graph-6

The second bar shows the number of logical records that are fragmented in RED. The fragmentation count is found by adding all the basic records that exist but cannot be combined into a logical record of the given type. This fragmentation count is then converted into logical records and shown as the value of the second bar. Essentially, the count given by the second bar is the number of logical records that could have been allocated had there been no fragmentation in the file at all.

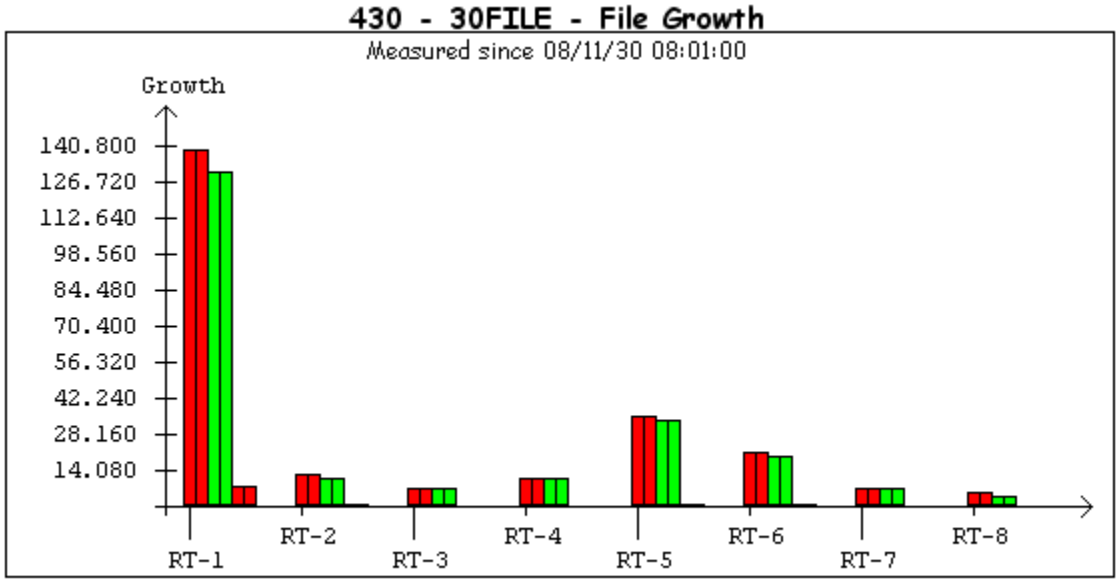
Note that fragmented records shown, for example, for record type 8 may be allocatable as they may be used for record allocation of a smaller record type if the D-value will permit such an allocation.

A large fragmentation for a given record type is naturally only a problem if this record type is actually used in the file. Use graphs 4 and 7 to get an idea of how heavily a given record type is used.

To view the actual count for a record type, click the individual bars.

Detailed File Analysis – Graph 7

The seventh graph shows the Freespace handlers allocation statistics for each record type. It displays the growth pattern along with the present record allocation and record releases within three times two (3x2) bars for each record type.



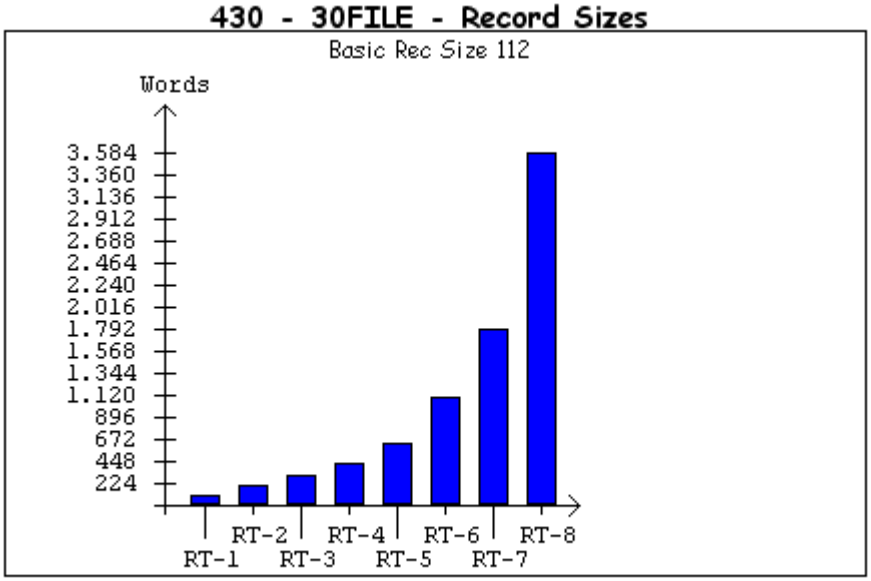
Graph-7

The first two bars show the number of record allocations since the statistics were last reset. The first of the two bars – when the graph is toggled to show percentages – displays the number of allocations in percentage of all the free records of this type. The second bar shows the number of allocations in percentage of the total amount of records of this type in the entire file. These two bars are always colored **RED** as they depict record allocations.

The next two bars will show the number of record releases since the statistics were last reset. These two bars are always colored **GREEN** as they show releases. The last two bars will show the net allocation value (in **RED**) or net release value (in **GREEN**).

Detailed File Analysis – Graph 8

The eighth graph very simply shows the setup of the individual records types in the file. Once the file is established, this graph will remain static as long as the setup remains unchanged.



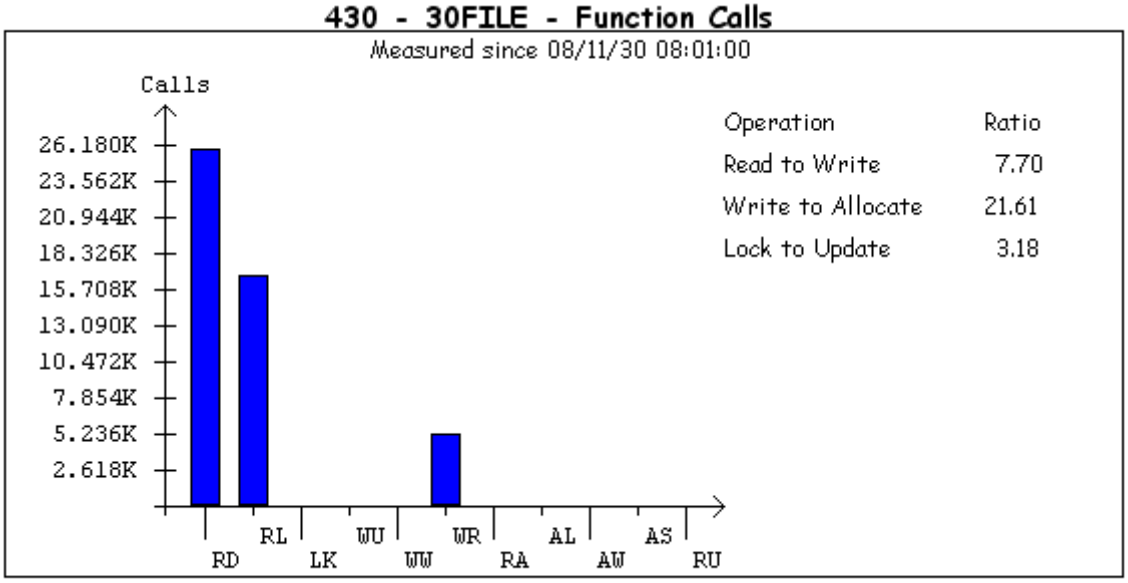
Graph-8

This graph displays the number of words that are used for each record type and provides a graphical picture of the area gained when switching from one record to the next.

If you double-click the white space of this graph, it will toggle between number of words and number of basic records used for each record type.

Detailed File Analysis – Graph 9

The following graph of the Detailed File Report describes how the records within a file are used by each FCSS function. Each file has its own unique pattern of FCSS function calls based upon the nature of the file and the data it contains.



Graph-9

This graph provides very useful information when, for example, designing data structures for the file or evaluating existing application programs that use the file.

The data within this graph includes ratios describing how the file is logically used.

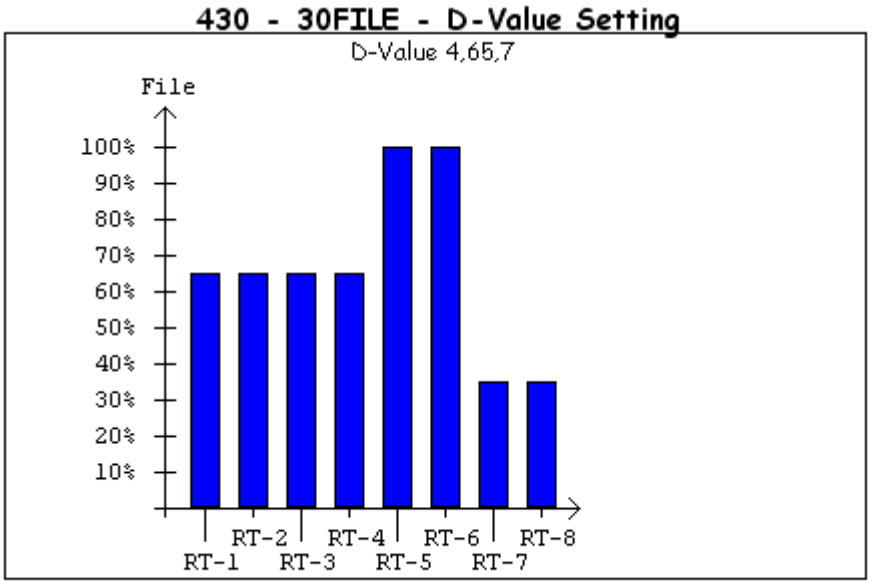
The **READ TO WRITE** ratio describes how often the data in the file is read compared to how often the data is written.

The **WRITE TO ALLOCATE** ratio describes how often a record is typically updated after it has been allocated.

The **LOCK TO UPDATE** ratio describes how often a record is read-with-lock compared to how often it is actually updated. This ratio should ideally be one with any value higher than this indicating that unnecessary record locks exist. Unnecessary record locks can often cause problems with application throughput, scalability, and general application performance. Thus, for heavily used files, this value should be kept as low as possible.

Detailed File Analysis – Graph 10

The tenth graph is a visual representation of the D-value. It shows how much of the file is available for record allocation for each record type.

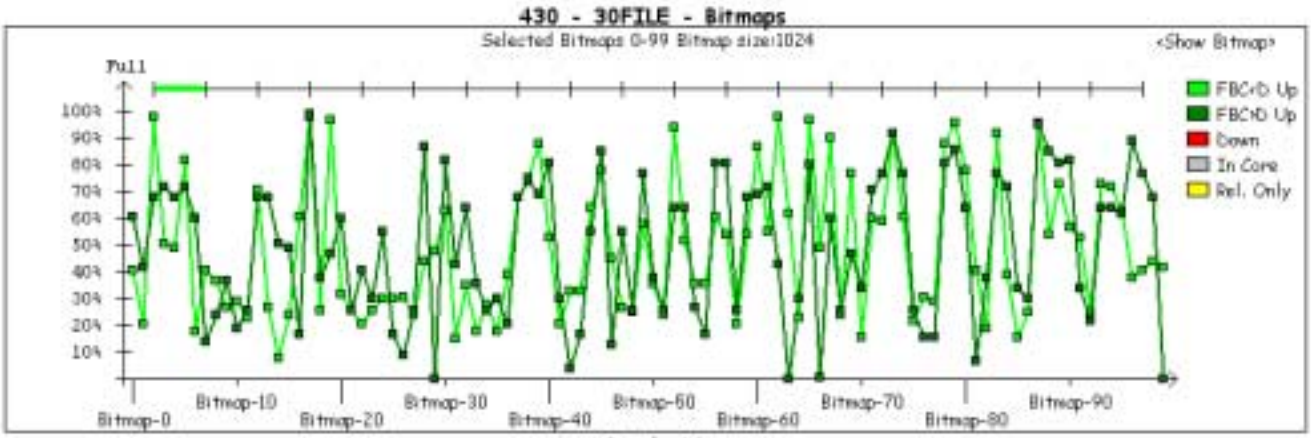


Graph-10

In this example, record types 1 through 4 are limited to 65% of the file. Because record types 5 and 6 are not specified in the D-value allocation, these records are not restricted by the D-value and thus can be allocated within 100% of the file. Finally, record types 7 and 8 are restricted to only 35% of the file.

Detailed File Analysis – Graph 11

The eleventh graph displays a file as seen from the perspective of individual BITMAPs. This unique view of a file shows the basic control information used by the Freespace Handler to find and allocate new records.

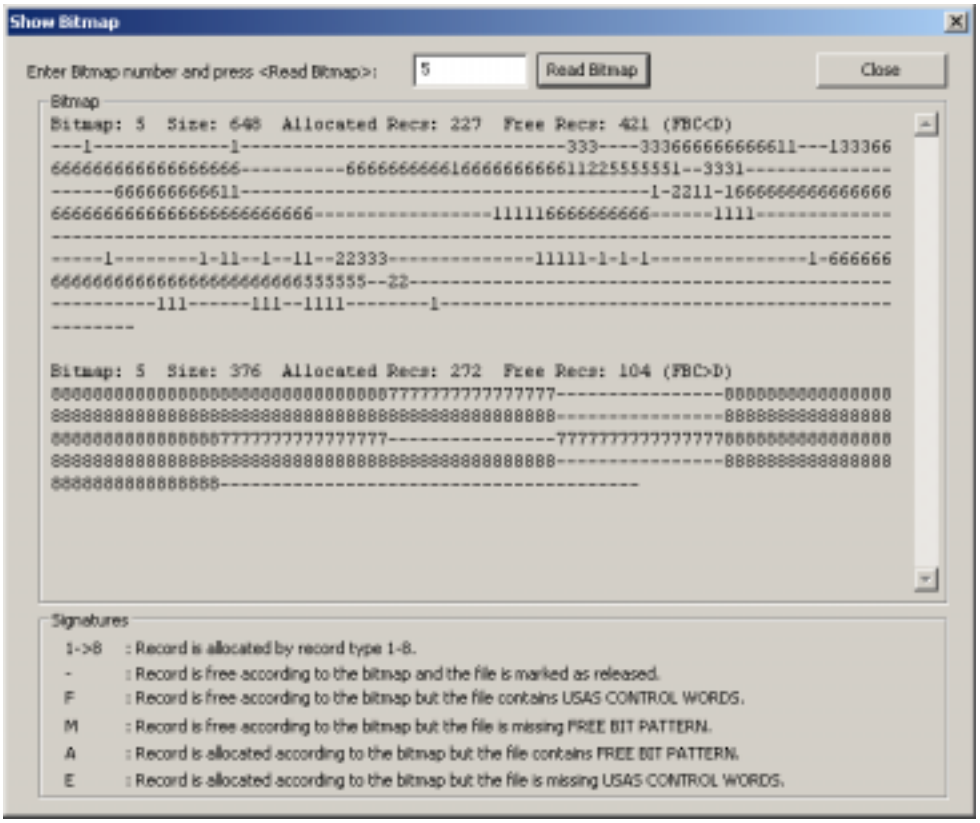


Graph-11

Because the number of BITMAPs in a file can be very large, this graph only shows a portion of the file at a time. To select a different portion of the file, click on the horizontal select bar at the top of the graph.

This graph shows how full each BITMAP is – both above and below the D-value – using a line graph for each area. The status of each BITMAP is color-coded. Click the line square boxes to view a yellow Pop-up window containing information on the actual values for the clicked BITMAP.

To get a more detailed view of a specific BITMAP, click **<Show Bitmap>** in the top left corner. This will open the below dialog.



Enter the number (zero relative) of the BITMAP you want to inspect, press “Read Bitmap” and the data related to this BITMAP will be displayed.

If the file contains a D-value, the BITMAP is split according to it to show how the records are allocated within the BITMAP. Each logical record is represented by its record type number (1->8) to show how the basic records are actually allocated. You can view this BITMAP display for a file while using the "What If D-value" feature to determine what the effects of a D-value change will be for a specific BITMAP. (*See **What If D-value***)

When the BITMAP analysis only includes numeric values, the BITMAP is fine as it matches the file and contains no discrepancies. If the analysis includes any letters – F, M, A, and E – there may be a problem with the BITMAP. The letters F and M are not usually a problem as long as the records aren't being used by the application. Because the records are considered to be free by FCSS, they will eventually be used for a new record allocation, which in turn will fix the problem.

The letter A may not actually be a problem as it indicates that a record has been allocated, but it was never written. This could be a valid situation for some applications, however, more likely than not it is an indication that the record has been "lost" to the system and is orphan. The next time a bitmap recovery is run, the record will be given back to FCSS and will be considered free.

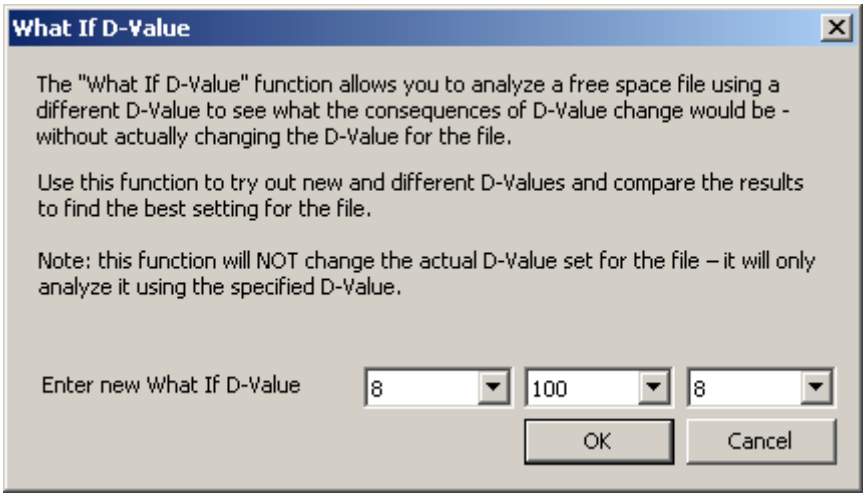
The letter E always indicates an error as these records are considered to be used by FCSS even though they can never be read by USAS. Thus, these records are lost for good. Moreover, because they do not hold the free bit pattern, they will not be recovered by a bitmap recovery. The only way to recover these records is to issue an unconditional release (XFSR) for each basic record flagged with the letter E. (See ***Freespace Scans***)

What-if D-value

The Visual Dump Analyzer allows you to produce a complete Detailed File Analysis Report using a different D-value than the one the file is currently using. The purpose of this is to allow you to see the consequences of a possible D-value change before it is actually updated. (See ***Detailed File Analysis*** above)

Rather than guessing on a new D-value hoping it will eliminate a fragmentation problem within a file, you can now determine with certainty if this is true or not. This analysis includes detail of the actual number of free records of each record type that will become available if the D-value is changed – hence the name What-If D-value.

To request a Detailed File Analysis using a new what-if D-value, click the Detailed Analysis Report graph for the file in question and select "**What if D-value**" on the Pop-up menu. Or, click the bar that represents the file in the File Report. This will open the below dialog allowing you to enter the "**What If D-value**":



Note that this will not change the D-value, but it will only generate a report about the file as if the D-value had been updated. To actually change the D-value, you will still need to use FREIPS.

Be aware that the D-value percentage used by the FCSS freespace handler must ”split” the words used for a BITMAP on word boundaries. If a specified D-value percentage does not calculate to a word boundary the D-value “split” is rounded so that it will.

The formula for calculating the BITMAP “split” is:

BITMAP_size_below_the_D-value = (BITMAP_SIZE * D-value_PCT / 100 + 18) / 36 * 36
BITMAP_size_above_the_D-value = BITMAP_SIZE - BITMAP_size_below_the_D-value

Consequently, a file with 1024 basic records per BITMAP will use 28.44 words to hold the BITMAP. If the file has a D-value of 7,50,8; the result will be 14 words (equivalent to 14*36=504 basic records or 49.2%) used for record types 1-7 and 14.44 words (equivalent to 14.44*36=520 basic records or 50.8%) for record type 8.

The following table shows the actual effect of setting various D-value percentages for a file with a BITMAP size of 1024 Basic Records per BITMAP.

D-value PCT	BITMAP SIZE<D	PCT<D	BITMAP SIZE>D	PCT>D
43	432	42.2	592	57.8
44	468	45.7	556	54.3
45	468	45.7	556	54.3
46	468	45.7	556	54.3
47	468	45.7	556	54.3
48	**504**	**49.2**	**520**	**50,8**
49	**504**	**49.2**	**520**	**50,8**
50	**504**	**49.2**	**520**	**50.8**
51	540	52.7	484	47.3
52	540	52.7	484	47.3
53	540	52.7	484	47.3
54	540	52.7	484	47.3
55	576	56.3	448	43.8

Table 1 – D-value splits on BITMAP size 1024

As you will note from this table, changing the D-value from 50% to 49% or 48% would not have any effect on the actual D-value used by the FCSS freespace file handler. The D-value would have to be changed to 47% to actually create an effect.

To simplify the selection of a new D-value, the **D-value PCT** column has been updated to only contain the percentage values that actually cause an updated D-value.

Freespace Scans

The Visual Dump Analyzer has its own Freespace Scan batch job called SEDFRS/BATCH. This program has two main features:

1) To scan all freespace files and save counts of records that are used for each subfile.
2) To verify the control information for freespace files and fix any discovered errors.

The first feature – to scan all freespace files and save counts – is started in HVTIP by entering the following:

RFIL:FSSCAN

This input will start The Visual Dump Analyzer Freespace Scan batch job, which will scan all the freespace files on the system and save the analysis in file USAS*SED$FSSCAN.

This input should be entered into Timecall and be run at a predetermined time each day. It will enable The Visual Dump Analyzer to produce the fifth graph in the Detailed File Analysis Report, which states how many records of each record type are allocated for a subfile in a shared file.

The Freespace Scan can also be started from demand by starting the batch job as stated below. Generally, however, scanning all the files once a day should be sufficient.

@START USAS*SED$FILE.FSSCAN

The second feature – to verify the control information for freespace files – can only be performed by running the SEDFRS/BATCH program in demand mode. This program has a built-in help screen to assist you with its use. (See ***The Visual Dump Analyzer – Scanning Freespace Files***)

Table of Contents

USER GUIDE FOR THE VISUAL DUMP ANALYZER **3**
OVERVIEW **3**
THE ENVIRONMENT **3**
SYMBOLIC DEBUGGING **3**
CONFIGURATION **4**
COMMUNICATION SETUP 4
GETTING STARTED – AUTO CONFIG 4
SEARCH PATH SETUP **5**
READING DUMPS FROM THE MAINFRAME **7**
DUMP ANALYSIS MODES **7**
QUICK MODE 7
NORMAL MODE 8
FULL MODE 8
FINDING THE ABSOLUTES AND SOURCE CODE **9**
READING OR REPLACING ABSOLUTES 9
READING OR REPLACING SOURCES 10
BROWSING PROGRAM FILES FOR ABSOLUTE OR SOURCES 10
VALIDITY COLOR CODES **12**
SAVING DUMPS TO DISC **13**
OPENING DUMPS SAVED ON DISC **13**
EMAILING DUMPS **13**
CUSTOMIZING COLORS AND FONTS **14**
CUSTOMIZING SETTINGS **15**
VIEWING THE DUMP **15**
RESIZING THE WINDOWS 15
SETTING THE ACTIVE WINDOW 16
SPLITTING THE WINDOWS 16
CHANGING WINDOW TYPES 16
THE SOURCE SELECTION TOOLBAR **16**
THE DATA SELECTION TOOLBAR **17**
THE GENERAL INFO WINDOW **17**
THE SOURCE CODE WINDOW **18**
SYMBOLIC DEBUGGING 18
POP-UP MENU FUNCTIONS 18
THE VARIABLE WINDOW **19**
THE EXPRESSION WINDOW **20**
THE DRAWING WINDOW **21**
THE PROGRAM CALL STACK (WALKBACK) 22
THE DATA DISPLAY WINDOW **23**
RESCHEDULING THE INPUT MESSAGE **25**
RESCHEDULING NORMAL TERMINAL TRANSACTIONS 26
RESCHEDULING HTH MESSAGES 26
RESCHEDULING SERVICES 28
EDITING OR FINDING THE INPUT MESSAGE 28
SCHEDULING EDIFACT MESSAGES DIRECTLY USING EDIT 29
SETTING TST TRACES 29
SETTING VISUAL ADVANCED SNAPS 29
CONTROLLING THE AP DBA 30
REMOVING TST TRACES AGAIN 30
SETTING VISUAL BREAKPOINT TRAPS **31**
SETTING TRAPS ON LINE NUMBERS 32
SETTING TRAPS ON MASM INSTRUCTIONS 33
SETTING TRAPS ON INTERNAL SUBROUTINES 34
SETTING TRAPS ON HVTIP CALLS 36
SETTING VISUAL BREAKPOINT TRAPS ON DATA **37**
SETTING TRAPS ON VARIABLES 38
SETTING TRAPS ON DBAS OR PDBS 40
SETTING TRAPS ON AN FIXED ADDRESS 41

FORCING ABORTS WHEN A MEMORY LOCATION GETS DESTROYED 41
SETTING VISUAL BREAKPOINT SNAPS 42
SETTING SNAPS ON VARIABLES 43
SETTING SNAPS ON DBAS, PDBS OR A FIXED ADDRESS 43
DYNAMIC PRTADP CALLS 44
VIEW CONSTRUCTION 45
AUTO-GENERATION OF *VIEWS* 46
THE VDA KNOWLEDGE DATABASE 47
REASON FOR ABORT TEXT 48
HINT TEXT 48
X11 HINT TEXT 48
ABORT SEVERITY 49
RESOLVE PROGRAMS 50
DEFINING RESOLVE PROGRAMS 50
STEP 1 – ASSOCIATING PROCS TO RESOLVE PROGRAMS 50
STEP 2 – UPDATING THE SEDUSR-F PROC 51
GENERATING RESOLVE PROGRAMS 51
ERROR CODE LOOK-UP 52
THE VISUAL DUMP ANALYZER LOG FILE 53
FORTRAN OPTIONS 53
BASIC MODE FORTRAN OPTIONS 53
EXTENDED MODE UCS OPTIONS 53

MONITORING USAS APPLICATIONS 54
ABORT REPORTS 54
READING ABORT REPORTS FROM THE MAINFRAME 54
ABORT REPORT TYPES 54
SHOWING ABORT REPORTS 54
SHOWING ABORT TOTALS 55
CLASSIFYING ABORTS AS IDENTICAL OR SIMILAR 56
SHOWING ABORT SEVERITY REPORT 56
SHOWING STATISTICAL REPORT 56
SHOWING SPO ALERTS 58
CUSTOMIZING THE ABORT REPORT LAYOUT 59
AUTO REFRESHING ABORT REPORTS 60
RAISING ALERTS MESSAGES 60
ALERT MESSAGE ANALYSIS 60
SUPPRESSING ALERT MESSAGES 62
CONFIGURING ALERT MESSAGE WARNING LEVELS 63
WARNING ON ALL ABORTS 63
WARNING ON TOO MANY IDENTICAL ABORTS 63
WARNING ON TOO MANY DIFFERENT ABORTS 64
WARNING ON REOCCURRING ABORTS 64
RUNNING IN MONITORING MODE 65
RAISING ALERT MESSAGES ON YOUR LOCAL WORKSTATION (IWS) 65
RAISING ALERT MESSAGES AS EMAIL OR SMS 66
RAISING ALERT MESSAGES AS REMEDY TICKETS 67
RAISING ALERT MESSAGES ON THE SPO 67
RUNNING VDA IN BACKUP MODE 68
SETTING REPORTS FILTERS 69
COMMAND LINE OPTIONS 70
SAVING ABORT REPORTS TO DISC 70
OPENING ABORT REPORTS SAVED ON DISC 70
EXPORTING ABORT REPORTS 70
EMAILING ABORT REPORTS 70
THE VDA ADDRESS BOOK 71
THE VDA PROGRAM BOOK 72

MONITORING FREESPACE FILES **73**
READING FILE REPORT FROM THE MAINFRAME **73**
FILE REPORTS **74**
FINDING FILES 75
SETTING FREESPACE FILES WARNING LEVELS 76
AUTO-REFRESHING FILE REPORTS **77**
RAISING ALERTS MESSAGES ON FREESPACE FILES **77**
ALERT MESSAGE ANALYSIS FOR FREESPACE FILES **78**
CONFIGURING ALERT MESSAGE WARNING LEVELS FOR FREESPACE FILES **79**
WARNING LEVELS FOR FREESPACE FILES 79
FREESPACE FILE SEVERITY CODES 80
ALARM METHOD 1 – EARLY WARNING 80
ALARM METHOD 2 – SIMPLE ALARMS 80
DETAILED FILE ANALYSIS **81**
DETAILED FILE ANALYSIS – GRAPH 1 81
DETAILED FILE ANALYSIS – GRAPH 2 82
DETAILED FILE ANALYSIS – GRAPH 3 83
DETAILED FILE ANALYSIS – GRAPH 4 83
DETAILED FILE ANALYSIS – GRAPH 5 84
DETAILED FILE ANALYSIS – GRAPH 6 85
DETAILED FILE ANALYSIS – GRAPH 7 86
DETAILED FILE ANALYSIS – GRAPH 8 87
DETAILED FILE ANALYSIS – GRAPH 9 87
DETAILED FILE ANALYSIS – GRAPH 10 88
DETAILED FILE ANALYSIS – GRAPH 11 89
WHAT-IF D-VALUE 90
FREESPACE SCANS 91
TABLE OF CONTENTS **93**