User Guide

for

THE VISUAL DUMP ANALYZER

Basic Mode

Version 5.0

To get 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

User Guide for The Visual Dump Analyzer Basic Mode

Reading Dumps (RDMP)

Introduction

The Visual Dump Analyzer Basic Mode compliments the The Visual Dump Analyzer GUI by providing you quick access to virtually all portions of a Unisys USAS dump file in an HVTIP environment. While doing this, it also performs extensive analysis of the dump by correlating its data with its associated absolutes and source code, thus eliminating the time-consuming and error-prone process of converting variables and lines of code to their absolute addresses within the dump.

Beyond quickly providing the necessary data that assists a programmer in analyzing a dump, The Visual Dump Analyzer has extensive built-in knowledge of the USAS environment that often points a programmer to the exact circumstance of the abort. Guard modes, substring errors, memory allocation errors, and many others point the programmer to the exact location of the error resulting in great time saving with this otherwiser error-prone process.

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 increases with time for critical details that are easily forgotten will always be available for automatic reference.

Finally, the Visual Dump Analyzer is able to resolve programs and dumps that exist in the USAS Mixed Mode environment. Regardless of whether the dump originates from batch, online batch or online programs – or whether they exist within Extended Mode or Basic Mode environments – the information is presented in the same easy-to-understand way giving you quick access to all of the relevant data.

The Visual Dump Analyzer environment

Flexibility has been designed into The Visual Dump Analyzer allowing you to customize your working environment. This includes defining search paths for finding absolute and source elements, modifying the displays by changing parameters in your personal setup, and allowing for various formats of output with the dump information requests.

The system and personal search paths

The Visual Dump Analyzer has two types of search paths: one for finding absolute elements and another for finding source elements. Each of these two search paths is composed of a *personal* search path and a *system* search path, with the *personal* search path being traversed before the *system* search path when looking for an absolute or source element.

There is a unique *system* search path and *personal* search path defined for each computer system. On production systems, generally a personal search path is not required as all elements are stored within system files. On development systems, however, you will need to add your own files to your *personal* search path on each computer system so that the correct absolute and source elements can be found.

To update your *personal* search path and display the *system* search path on a computer system, just enter:

RDMP:PATH

Your *personal* search path is saved in the file **SED$*nnnnn.** where nnnnn is your VDU number (i.e., USAS PID number). Therefore, you will need to update your search path on all the PIDs you are using.

Element search methodology

When searching for an absolute element, The Visual Dump Analyzer will first examine the HVTIP Program Bank Library to determine which file the online absolute element was loaded from. Next, this file will be examined to determine if the exact (i.e., having the same date/time stamp) absolute element exists in this file. Moreover, if the absolute is loaded from TPF$, this step will be skipped as temporary files are unable to be accessed.

If the exact absolute element is not found, The Visual Dump Analyzer will next search the files in the *personal* and *system* search paths. If an exact match still cannot be found, the absolute element that *best* matches the date/time stamp in all the files searched will be used. Furthermore, 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.

When a batch program creates a dump, The Visual Dump Analyzer will automatically capture the batch absolute element at the time of the abort and save it in the dump file. This ensures that the correct absolute element will always be available for batch dumps, even if it is executed out of TPF$ file.

Your personal setup

You can customize the appearance of the RDMP: displays for your personal setup by modifying the configurable parameters in The Visual Dump Analyzer. To modify these settings, enter:

RDMP:SETUP

A couple of items are worth noting in the personal settings. First, when `DISPLAY OCTAL AND ASCII SIDE BY SIDE` is set, the data is displayed with four (4) octal words on a line and the ascii representation to the right. When unset, the traditional five (5) octal words on a line with its ascii representation displayed below is used.

Next, when `DISPLAY SOURCE CODE WHEN POSSIBLE` is set, errors that can be further described by displaying its related source code will result in ten (10) lines of code in the output with the line of source code most related to the error being centered.

Your parameter settings are saved in the file **SED$*nnnnn.** where nnnnn is your PID number. Therefore, you will need to update your setup on all the PIDs you are using. Note that this is the same file that stores your personal search path.

Requesting information from the dump

The basic format for retrieving data from the dump first includes the RDMP: transaction and the dumpfile$d file name. Next, a KEYWORD can be provided along with additional parameters refining the keyword selection. The keyword specifies the actual information that is desired from the dump file. For keywords that display data areas within the dump, an offset parameter can also be included stating the word to begin displaying from. Continuing, a format parameter can be defined in order to adjust the display's data. Finally, the INFO clause can be entered to display the date/time stamps of associated absolute elements and the RAW clause can be specified to view the data in a purely octal format.

RDMP:dumpfile$d KEYWORD [/add-info/add-info…] [/offset] [/format] [/RAW] [/INFO]

Whenever the analysis of the dump derives addresses within the I-bank of the dump, The Visual Dump Analyzer will attempt to convert the absolute address to a source element and line number. If the element is a masm program, the line number conversion is not possible and the output will state the relative address within the relocatable.

Note: clauses that are optional in an input string are always placed in brackets (i.e., [optional]).

Formats of the displayed data

The Visual Dump Analyzer displays data in its native format so it is most recognizable. For example, **STACK** displays are shown with symbolic names of the source code variables along with their associated values. If you wish to view the data being requested in a raw data format, simply include the ***RAW*** clause anywhere in the input line.

When The Visual Dump Analyzer displays information in a raw data format, you may select the format of it by entering the ***format*** clause. Values that may be defined for this clause include: **Octal, Ascii, Decimal,** and **Edifact.** Moreover, the values can be abbreviated to their first character (i.e., **O**, **A**, **D** and **E**). The ***format*** clause can appear anywhere in the input string.

Using the INFO clause to verify elements

In order for The Visual Dump Analyzer to perform automated analysis of a dump and for you to examine a dump using symbolic debugging, it is important that The Visual Dump Analyzer is able to find the exact absolute and source elements that relate to it. Without the correct elements, the analysis can easily be flawed resulting in misleading or incorrect solutions.

Therefore, it is always a good idea for you to verify that the elements being used to analyze a dump are the correct ones. The Visual Dump Analyzer will still analyze a dump and allow you to examine it when the elements are incorrect, but it will inform you with warnings stating which parts of the analysis and output are possibly in error.

To view the absolute elements that are being used for the dump analysis, enter an ***INFO*** clause anywhere within the input string. A simple example of this input follows:

RDMP:*dumpfile$d D INFO*

The file from which the absolute elements were retrieved from along with the date and time the absolutes were created will be included in the display.

The Initial RDMP: Display

The Initial RDMP: Display provides an overview of the dump by including a header containing general environment data, the input message, a list of allocated DBAs and located PDBs, an analysis of the present error codes, and a quick view of the location of errors in the source code, amongst other information. The Visual Dump Analyzer accomplishes this by scanning the entire dump, correlating its contents with the associated absolute and source elements, and using its extensive built-in understanding of the USAS environment. During this analysis, The Visual Dump Analyzer will also present you with areas that are deemed to be of specific interest and are thus more likely to be involved in causing the abort. This means that the Initial RDMP: Display may differ with each dump. To request the Initial RDMP: Display, enter:

RDMP:*dumpfile$d*

The Initial RDMP Display also includes information retrieved from the Knowledge Database. Depending upon the errors, this may include a *Reason for abort text* or *Hint text*. See *The Visual Dump Analyzer Knowledge Database* for more details.

The Visual Dump Analyzer Header

All displays from The Visual Dump Analyzer contain a fixed header showing general information such as dump name, date/time the dump occurred, basic error code information, and an echo of the last input string. An example of the header follows:

```
RDMP:SEDDMP$D
********************************************************************************
* TIP$*SEDDMP$D(1).    PRG:SEDDMP    RUN:*26DLP    IBSERR:572      SITEID:TDEV-E *
* 05/24/01 09:09:58    FNC:RDMP      VAL:MINI1     IBDIAG:5276     IBACBN:XBCA   *
* CONT:000000000000    PID:3822      AIR:SK        IBNOTE:73059    IBACBR:037255 *
********************************************************************************
```

Requesting specific information from the dump

Besides the Initial Display where The Visual Dump Analyzer analyses the dump and selects the areas it deems to be of most interest, you can also request specific areas to be translated and displayed. The following section will describe these requests and the data they present.

Requesting data from a specific address

In order to display data in the dump beginning at a specified address, enter:

RDMP:*dumpfile$d address [loffset] [lformat]*

The ***address*** you specify can be either a numeric address with a leading zero indicating octal, a common block name, fixed text "PCT", AXR$ register, an entry point name or a relocatable area in the d-bank. Essentially, any input that can be converted into an address can be specified. When a specific data area is specified, The Visual Dump Analyzer determines its length and only displays its data.

STACK requests

By displaying the names of the variables in the source code and their associated values, the STACK request depicts the program stack area - sometimes referred to as the Program Work Area. The variables are displayed in alphabetic order for each relocatable subroutine mapped into the absolute, with each subroutine being further split into their internal subroutines. The variable information includes data type, stack relative address, absolute address and value.

If you wish to display all the variables within the active program stack area – that is the stack belonging to the aborting program, enter the following:

RDMP:*dumpfile$d* STACK

If you wish to view a program stack belonging to an absolute earlier in the call-sequence than the aborting absolute, simply specify the name of this **absolute element** as follows:

RDMP:*dumpfile$d* STACK / absolute-elt

The output for a STACK request is based upon the dump's Walkback (i.e., Program Call Stack) information. The Walkback information contains the sequence of *internal* subroutines that would be *returned to* if processing had continued and returned to its origin. It is *not*, however, a trace of all subroutines that have been traversed during a transaction's complete processing.

The output for this request, then, contains these internal subroutine stack areas, which are shown in reverse-order with the active (aborting) subroutine's stack being listed first. Furthermore, this display also presents the valuable information of the *primary path* taken within the program in arriving at the point of the abort. This path does not, however, include "side trips" to subroutines that have already been returned from as it is not a *trace*, but instead a *snapshot* of memory at the time of the abort.

It often occurs that other stack areas exist within the dump, but they are not valid because they have either returned to their caller or have never been called during execution. In circumstances where the stack areas were once active but have returned, the contents of the stack areas may be of interest.

To request all stack areas in a dump including those that are not within the Walkback information, enter:

RDMP:*dumpfile$d* STACK / ALL … or
RDMP:*dumpfile$d* STACK / absolute-elt / ALL

This will result in a list of stack areas beginning with the main program's stack followed by *all* the stack areas in the order of appearance within the dump without regard to the Walkback information.

In the normal stack display, only the first characters are shown for long tables and strings. If you wish to view an entire table or string in its textual and raw data format, enter the **variable** name in the following format:

RDMP:*dumpfile$d* STACK *[/absolute-elt] / variable [/format] [/*ALL*]*

If the ***variable*** is in more than one relocatable or internal subroutine, each occurrence will be displayed. Further, if ***absolute-elt*** is omitted the variable specified is shown from the stack of the aborting program. Finally, if this is not the stack you which to obtain the variable from, you must supply the ***absolute-elt*** to inform which stack to use.

When you are in doubt as to what stacks exist in the dump, you can use the following Program Call Stack (Walkback) request to obtain information on all the stacks present in the dump.

Program Call Stack (Walkback) requests

This display 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 displayed where each program calls forward to an internal subroutine, a mapped in relocatable, an ACB routine or another HVTIP program. Each of these calls that inform on the location of the forward call is displayed on a separate line. For Fortran programs, the location is a source line number in decimal and with masm programs it is an absolute address in octal. These locations, which are placed in parenthesis at the end of each line, can be used to determine the path that processing took before arriving in the aborting subroutine.

The Program Call Stack (Walkback information) is requested by entering:

RDMP:*dumpfile$d* CALLS

The information provided by the Walkback display reveals the path that processing took in getting to the abort location. Moreover, this data should be considered an analysis of the program stack areas and not a detailed trace of the program's execution.

To receive an overview of all the program stacks that are defined within the dump, enter:

RDMP:*dumpfile$d* CALLS/RAW

This display will show each program stack header in its raw format along with translation of the header data, but without performing the Walkback analysis. The translated data includes the HVTIP library/bank, program/relocatable name, stack address and stack size in words.

Input and Output Message requests

The Initial RDMP: Display includes the input message in textual form. You can also request the input message with a different format by entering the following:

RDMP:*dumpfile$d* IMSG *[/format]*

If, instead, you wish to view the input message along with its associated control information (i.e., the Message Parameter Area), the request is as follows:

RDMP:*dumpfile$d* MPAI *[/offset] [/format]*

Similarly, if you wish to view the output message, enter:

RDMP:*dumpfile$d* OMSG *[/format]*

And, finally, to view the output message along with its associated MPA area, enter:

RDMP:*dumpfile$d* MPAO *[/offset] [/format]*

PDB requests

An overview of the located and locked PDB tables at the time of the abort can be requested with the following input:

RDMP:*dumpfile$d* PDBS

The located and locked PDBs will be displayed separately, with the output for each set being in ascending address order.

To view the raw data of a specific PDB table, use the following input:

RDMP:*dumpfile$d* PDB / *pdb-name [loffset] [lformat]*

Finally, to view a PDB table in a symbolic-oriented and more informative way, please refer to the sections *View Construction* and *Resolve Programs*.

DBA / Memory Usage requests

This request provides a thorough overview of the use of dynamic memory at the time of the abort. This includes allocated DBAs, locked PDBs, program stack areas and unassigned (i.e., free) blocks of dynamic memory. The general request is as follows:

RDMP:*dumpfile$d* DBAS

Each DBA and locked PDB is named by its (often two-character) system mnemonic, program stack areas are named RESV, unassigned dynamic memory are named FREE and other areas reserved for system-related usage are also named RESV. For ease of analysis, all the dynamic memory areas are shown in ascending address order.

In batch dumps, the dynamic allocation of memory is controlled by the Storage Management System (SMS). In order to map out dynamic memory usage in batch dumps, The Visual Dump Analyzer will examine SMS's current allocate block and free chain. Batch dumps, however, have many dynamic areas actually allocated by FTN to perform functions such as SDF file I/O. All these system-related areas are identified as reserved (i.e., RESV).

You can toggle the inclusion of free and reserved areas in this display within your personal setup (i.e., RDMP:SETUP).

To view the contents of a specific DBA, use the following format:

RDMP:*dumpfile$d* DBA / *dba-name [loffset] [lformat]*

Some DBAs have a known layout and thus The Visual Dump Analyzer will show these DBAs translating their data into a more readable format. Ex the PG DBA will be shown as a screen output – showing the last part of the output screen produced.

RDMP:*dumpfile$d* DBA / PG

In the same way the AP (the trace dba) will be shown giving the last part of the TRACE data produced.

RDMP:*dumpfile$d* DBA / AP

To prevent The Visual Dump Analyzer to translate these DBAS simply use the RAW keyword and the DBAs will be shown in their RAW formats.

Finally, to view a DBA in a symbolic-oriented and more informative way, please refer to the sections *View Construction* and *Resolve Programs*.

BCB / Buffer Control Blocks

The BCB keyword functions just like the DBA keyword only it informs The Visual Dump Analyzer that the allocated DBA is preceded with at BCB (Buffer Control Block). This display of the DBA memory will be broken up into the 6 words BCB area followed by record data.

Further, if the record belongs to a Freespace file The Visual Dump Analyzer will translate the 3 USAS control word in the record to natural language text explaining who wrote the record last and when. The request is as follows:

RDMP:*dumpfile$d* BCB / *dba-name*

An additional benefit of this is that the relative word number shown is now matching the actual record data, rather than being off by the 6 words used for the BCB.

The Jump Stack requests

The Jump Stack is only present with contingency aborts because the EXEC only captures the jump stack with a specific set of interrupts. XTCA aborts do not initiate such an interrupt, thus the jump stack cannot be written to the dump file. To display the jump stack with contingency aborts, enter:

RDMP:*dumpfile$d* JSTK *[/RAW]*

The Visual Dump Analyzer will convert each entry in the jump stack to a program name and, if possible, to a line number. Further, the description of each entry is described with both the location and destination of the jump. For verification purposes, the absolutes are checked to ensure a jump actually exists at each location. If not, a message will be output warning that the absolute element used to analyze the dump is not correct.

Adding the ***RAW*** clause to the request prevents The Visual Dump Analyzer from editing the jump stack, in which case it will be shown as raw data.

Registers requests

To view the values stored in the AXR$ register set or the Extended Mode Base register set, enter:

RDMP:*dumpfile$d* REGS

This will display the contents of all registers in octal format with the registers identified and grouped by type with a space inserted between their half-words for ease of inspection.

If the dump is an Extended Mode dump, this display will also contain the Extended Mode Base Registers informing on the bank based at dump time.

Free Chain requests

If you wish to get a complete overview of the unallocated (i.e., free) dynamic memory at the time of the abort, simply enter:

RDMP:*dumpfile$d* FCHAIN

The Visual Dump Analyzer will traverse dynamic memory's free chain and verify that the free block token (i.e., octal 0444444) is present at appropriate locations indicating that the free chain is intact. If the free chain is found to be broken, the backward links will be traversed to determine the extent of corruption. Thus, if the free chain is only broken in one location, the valid chain can still be displayed.

Location, size, forward/backward links and chain status will be displayed for each free area recorded by the free chain.

When a dump is being analyzed that is the result of an aborting Extended Mode program, the Extended Mode free chain is traversed and its information is presented in the same way as with the Basic Mode free chains.

Because the Storage Management System (SMS) is used to control dynamic memory for Basic Mode batch programs, the normal CBBLOC Free Chain does not exist. Instead, The Visual Dump Analyzer will use the SMS current allocate block and the SMS free chain to produce a display similar to that created for online programs. Further, the total amount of free memory available for allocation by SMS is calculated and shown to inform of the largest buffer allocation still possible.

Last ACB call requests

The program name and, if possible, line number of the last ACB call can be displayed by entering the following request:

RDMP:*dumpfile$d* LASTACB

If you are using The Visual Dump Analyzer on a system where the source code is present, ten lines of source code will be displayed about the last ACB call providing you with a quick view of it.

FCSS I/O information requests

When an FCSS I/O results in an abnormal status, the error status is stored in the IBBLOC. When FCSS status information exists within the dump, it will be displayed in the Initial RDMP: Display. It can also be requested with the following:

RDMP:*dumpfile$d* FCSS

The location of the FCSS error is converted to a program name and, if possible, a line number. If you are using The Visual Dump Analyzer on a system where the source code is present, ten lines of source code will be displayed about the abnormal FCSS call.

You can also use this function to convert a FCSS system error code to its associated textual description by using the request:

RDMP:*[dumpfile$d]* FCSS / *fcss-error-code*

Note that ***dumpfile$d*** is optional when using this function.

Contingency information requests

When a contingency occurs that results in the creation of a dump, the contingency error is stored in the FSTACK$ area. Besides being displayed with the Initial RDMP: Display, the contingency can be converted into a textual description using the following:

RDMP:*dumpfile$d* CONT

The location of the contingency error is converted to a program name and, if possible, a line number. If the contingency occurred in an ACB – main or alternate - it too will be identified and used to find the source code causing the contingency. If the program is not Fortran and thus no line numbers exist, the location of the contingency will be displayed as a relative address in the aborting relocatable.

If the contingency occurs outside USAS in a BDI unknown to The Visual Dump Analyzer, the location of the contingency will be described as a BDI-value and absolute address.

If you are using The Visual Dump Analyzer on a system where the source code is present, ten lines of source code will be displayed about the contingency location providing you with an overview of it.

You can also use this function to convert a contingency code to its associated textual description by using the request:

RDMP:*[dumpfile$d]* CONT / *contingency-code*

Note that ***dumpfile$d*** is optional when using this function.

XTCA information requests

For XTCA aborts, The Visual Dump Analyzer will use the IBSERR value to reference its Knowledge Database to determine if further information exists for this abort type. That is, if any programmer solved a similar dump in the past and recorded a *Reason for abort*, *Hint text* and/or *X11 text*, this knowledge is automatically evaluated and displayed. (See *The Visual Dump Analyzer Knowledge Database*.)

Besides being displayed with the Initial RDMP: Display, the XTCA information can be retrieved with the following request:

RDMP:*dumpfile$d* XTCA

The location where an XTCA error has occurred is converted into a program name and, if possible, a line number. In addition to this, The Visual Dump Analyzer will attempt to step backwards in the processing. That is, it will try to determine what caused the call to the XTCA abort by using the available debug information. A result of this analysis is, for example, when an XTCA error occurs in an ACB routine and the XTCA location is resolved as the actual call to the ACB routine rather than within the ACB itself.

If you are using The Visual Dump Analyzer on a system where the source code is present, ten lines of source code will be displayed about the XTCA location providing you with a quick view of it.

You can also use this function to retrieve information from The Visual Dump Analyzer's Knowledge Database by manually entering an XTCA error code in the following request:

RDMP:*[dumpfile$d]* XTCA / *xtca-error-code*

This entry can also be used to test or verify new entries created in The Visual Dump Analyzer Knowledge Database. Note that ***dumpfile$d*** is optional when using this function.

TUXEDO (OLTP) information requests

When a Tuxedo error occurs, the error is stored in the USAS dump thus allowing it to be retrieved by The Visual Dump Analyzer. A textual description of this Tuxedo error is included in the Initial RDMP: Display and can also be directly requested with the following:

RDMP:*dumpfile$d* TUXEDO

You can also use The Visual Dump Analyzer to look up Tuxedo (OLTP) error codes by entering the TUXEDO error code manually as follows:

RDMP:*[dumpfile$d]* TUXEDO / *tuxedo-error-code*

The result of this error look-up is the standard Tuxedo error name, which is usually very short but necessary to reference a more descriptive text in the Tuxedo and OLTP manuals. For more common Tuxedo errors in the USAS environment, The Visual Dump Analyzer displays its own evaluation text that is more informative.

Edifact information requests

Edifact error analysis is generated from the control information that is stored in the EY-dba header, which includes the EYERRR value. This data is combined with the Edifact message structure as stored in the EZ-record and built-in knowledge of the Edifact functionality to provide an accurate and detailed description of the location and reason for the error occurring. Edifact error information is requested as follows:

RDMP:dumpfile$d EDI

An Edifact error often occurs during unpacking or construction of a segment. In this case, The Visual Dump Analyzer will detail the symbolic names and contents of the data elements within the active segment allowing for convenient analysis for the error at hand.

Using the following request, this function can also be used to look up EYERRR error values for both the Edifact input and output handlers:

RDMP: *[dumpfile$d]* EDI / *EYERRR-error-code*

Note that dumpfile$d is optional when using this function.

VIEW requests

Common blocks, DBAs, PDBs and DMS records are normally displayed as raw data, but you can also choose to display this data as edited displays, or *views*. Views are a powerful and flexible way of displaying data by incorporating freeform text and symbolic data from the dump and absolute elements resulting in a helpful natural language description. To display the contents of a record by using a View, simply enter:

RDMP:*dumpfile$d* VIEW / *view-name*

Before you can use a View, it must first be constructed. Views can be automatically generated from both $F and –F procs, or be developed manually for specialized use. (See *View Construction* in the *User Guide for The Visual Dump Analyzer GUI*).

If the view was constructed so it uses a POINTER or an INDEX variable as part of the view construction, this variable will automatically be looked up in the active stack and used to construct the view. However, to display the view using a manually set value for the POINTER or INDEX variable - use the below format:

RDMP:*dumpfile$d* VIEW / *view-name* / *var=expression*

This method allows you to control the construction of the view and the data is shows.

VALUE requests

This function allows you to request the value of an ***expression*.** An *expression* can be as simple as the name of a define, constant or variable in the stack (or in a DMS record), or it can be a complex mathematical expression containing any combination of these. To request the value of an *expression*, enter the following:

RDMP:*dumpfile$d* VALUE / *expression*

The result of an *expression* is displayed as octal, decimal, ASCII, fieldata and hexadecimal. Further, if the *expression* is a table or string, a raw dump of the entire table or string is also displayed.

In order to resolve an *expression*, The Visual Dump Analyzer uses the dump, active absolute and its inherent understanding of system procs along with the knowledge it has been taught about application procs through use of *Resolve Programs*.

Therefore, besides the variables from the program stack, *expressions* may also include variables defined in SPSYS and SYS common blocks (i.e., system variables prefixed with S$, CA, IB, CB, CQ, EY and TA, among others). Furthermore, all DMS record variables from all of the schemas and sub-schemas are also known.

You can also *teach* The Visual Dump Analyzer any variables that exist within an application proc by creating *Resolve Programs*. Resolve Programs are automatically constructed from either $F or –F procs. (See *Resolve Programs* in the *User Guide for The Visual Dump Analyzer*.)

The following are examples of *expressions* that can be 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.

This feature provides an easy way to view the values of variables in a dump. Be aware that if strings are used as part of mathematical expressions, only the first 36 bits of the string are used in the calculation.

When resolving *expressions*, The Visual Dump Analyzer uses the active stack by default. If you wish to override this and use another stack within the dump, use the following request:

RDMP:*dumpfile$d* *absolute-elt* / *relocatable* / *internal-sub* / VALUE / *expression*

Be aware that everything following the **VALUE** clause will be considered to be part of the *expression*.

FTN line information requests

The Visual Dump Analyzer will convert an address from an absolute element into the program name and line number that created the specified address by entering the following:

RDMP:*dumpfile$d* FTN / *address*

If the address you specify is not unique in that it may belong to a batch dump with multiple I-Banks, you can request the BDI (Bank) that should be used for the calculation as follows:

RDMP:*dumpfile$d* FTN / BDI## / *address*

Note that if you are unsure of which BDIs are defined for your banks, use RABS: with the BANK clause to determine all the BDI values for the banks.

Viewing I-Banks

To view the I-Bank from a dump (i.e., not from an absolute element), you can request The Visual Dump Analyzer to decode the dump's I-Bank and present it in a format specifying the absolute element and relative addresses along with the relocatable and internal subroutine names. The request is as follows:

RDMP:*dumpfile$d* *line-number*

The enhanced output appears Fortran-like with source line numbers, labels, variable usage, goto-statements and call-statements being shown with their symbolic names. This feature is especially helpful for non-masm programmers so they can more easily follow the generated masm code and be able to compare it to their Fortran source code.

DMS requests

The Visual Dump Analyzer's ability to read and understand DMS DDLs allows you to symbolically analyze the contents of DMS records. That is, you can reference all DMS records by *name* without knowing their DMS record number, location or size. Further, The Visual Dump Analyzer allows for multiple schemas/sub-schemas at a site by inspecting the DMCA area to determine which ones are active at the time of the abort.

To receive a quick overview of the DMCA area in an edited format along with a raw display of the active DMS record at the time of the abort, just enter:

RDMP:*dumpfile$d* DMS

If the DMCA reveals a DMS error the DMR basic error code is automatically looked up and the error information is given. In the same way DMR rollback error code is looked up and the error information is presented for easy viewing.

Besides the error codes given by the DMCA you can look up any DMR basic error code or DMS rollback error code by specifying the error code directly.

RDMP:DMR / *dmr-basic-error-code*
RDMP:DMRRB / *dmr-rollback-error-code*

This will cause The Visual Dump Analyzer to look up the error code and display the related error information.

By supplying the name of a DMS record, The Visual Dump Analyzer will find its location and size, and provide you with a raw display of the record. Simply enter:

RDMP:*dumpfile$d* DMS / *dms-record-name*

Because The Visual Dump Analyzer is aware of all DMS record variables within all schemas/sub-schemas at a site, you can use the VALUE request to display the contents of a variable in the following manner:

RDMP:*dumpfile$d* VALUE / *dms-variable-name*

Finally, in the same way that DBAs and PDBs can be viewed in an edited format, DMS records can also be displayed in a more informative way by using *views*. However, because all DMS records and their data fields are known to The Visual Dump Analyzer, you do not have to build resolve programs for them.

Help requests

You can request online help to assist you in formatting requests for The Visual Dump Analyzer by entering the following:

RDMP:*dumpfile$d* HELP

This will generate a list of all the possible inputs for this dump with the name of the dump inserted in each input line and tab stops allowing you to quickly move to the input you are interested in.

To view a more general help file, use the following **HELP** request to display *The Visual Dump Analyzer Basic Mode Command Sheet*:

RDMP:HELP

Quick Redisplays

You can request the last RDMP: display to be redisplayed directly from the paging file rather than reproduced by The Visual Dump Analyzer. To do so just enter RDMP: with any further information:

RDMP:

This will display the last RDMP: display, if one exists in one of the 5 paging areas. The benefit of this is that you can quickly re-obtain a RDMP: display from a dump without having to remember and key in the input string again.

As all The Visual Dump Analyzer function codes (RDMP: RABS: RSRC: RAPG: TRCE:) uses different paging levels you can use this logic to quickly toggle you display between RDMP: and any RABS: or TRCE: inputs using during your debugging.

X11 Hint control

Once an X11 Hint is updated for a given dump type (see *The Visual Dump Analyzer's Knowledge Database* later in this document) the X11 Hint will automatically be used in the analysis for all dumps of the specific type. However, you can control the use of X11 Hints manually by either disabling the X11 Hint or by replacing the X11 Hint with a new manually entered hint.

Disable an existing X11 Hint

To disable an X11 Hint you must apply the **NOHINT** keyword to the input string.

RDMP:*dumpfile$d* NOHINT

You can combine the **NOHINT** keyword with all other keywords that may base their logic on an X11 Hint. Entries that may base their logic on the presence of an X11 Hint are **CALLS**, **STACK** and the automated analysis. In all other cases the **NOHINT** keyword is still legal but will have no effect. Use this keyword to see what effect an X11 Hint has on a specific dump and to verify that the X11 Hint is actually valid or useful for the dump in questing.

Setting a manual X11 Hint

You can always set an X11 Hint manually – if the dump is already updated whit at X11 Hint in the Knowledge Base the X11 Hint from the knowledge Base is replaced with the manually X11 Hint.

RDMP:*dumpfile$d* X11=expression

The manual X11 Hint can be combined with the same kind of keywords as the **NOHINT** keyword can. In all other cases setting of a manual X11 Hint is still valid but will have not effect. Be aware that when the manual X11 Hint is combined with other keywords it must be entered LAST in the input string as it is entered in the form of an expression and therefore may contains slashed (divisions).

Even though The Visual Dump Analyzer's Knowledge Database cannot be updated with hints for contingency aborts you can set manual X11 Hints for contingencies. This is important as setting a manual X11 Hint for contingencies is the only way to inform The Visual Dump Analyzer of the true value of X11 if the contingency occurs in a MASM programs that has destroyed or reused the X11 register.

Requesting specific Extended Mode Information from the dump

In addition to the specific data that can be requested from both Basic and Extended Mode dumps, a range of requests exist that will return information within areas or data that relates *only* to Extended Mode. The following section describes these Extended Mode requests.

Requesting data from a specific Extended Mode bank

Because Extended Mode banks cannot be identified by their address alone, you must also specify the bank (BDI) along with the address to uniquely identify the data you wish to display. To do this, enter:

RDMP:*dumpfile$d* BDI## / *address [/format]*

The **BDI##** contains the fixed text "BDI" followed by a 1 to 4 digit decimal or octal (leading zero) bdi number. The level (L) of the bank (L,bdi) is automatically determined, so this should not be specified.

The Visual Dump Analyzer handles large banks – those exceeding 0777777 octal words in size – in the same way as smaller banks even though they are technically handled much differently by the Exec. Without having concern for the bank size, this results in an identical interface to all Extended Mode banks.

Once the bank is identified, you must specify the ***address*** as a 1 to 8-digit number, with a leading zero indicating octal.

Extended Mode DBA / Memory Usage requests

DBAs allocated within the Extended Mode AWA – the dynamic memory defined by the EMDBANK common bank – are viewed in the same way as Basic Mode DBAs. The following request provides an overview of the Extended Mode dynamic memory usage at the time of the abort:

RDMP:*dumpfile$d* EMDBAS

Extended Mode DBAs are named by their (often two-character) system mnemonic, unassigned dynamic memory blocks are named FREE, and other areas reserved for system-related usage are named RESV. As with Basic Mode DBAs, all the dynamic memory areas are shown in ascending address order.

To view the contents of a specific Extended Mode DBA, use the following format:

RDMP:*dumpfile$d* EMDBA / *dba-name [/offset] [/format]*

The ***offset*** and/or ***format*** parameters can be used to modify the way the data is displayed.

Extended Mode HEAPS requests

The HEAPS request will find and translate data contained within the Control Heap of the Heap Manager. This includes information describing the allocated $DSEG data segments, their owner programs, and the relocation of these segments. Information on all Extended Mode named or indexed common blocks and their location is also presented along with the Heap Manager free chain (not to be confused with the Extended Mode's dynamic area USAS free chain). To request this data, simply enter:

RDMP:*dumpfile$d* HEAPS

This request is useful to programmers that are debugging within the USAS Mixed Mode environment. Programmers debugging Extended Mode programs, however, will likely find more use with the specific heap displays that are used to display the $DSEG of a specific program.

To display the raw data and location of a specific program's $DSEG, use the following request:

RDMP:*dumpfile$d* HEAP *[/program] [/format]*

This request will break the $DSEG into the individual program data banks that make up the program's $DSEG. In the same way that $DSEGs for HVTIP programs are relocated at runtime, The Visual Dump Analyzer will not only inform you of the address range given to the individual data banks by the Linker, but also of the runtime relocated address range, which is the actual address range occupied by the $DSEG in the heap bank during runtime.

To view the raw heap data from a given address within the Control Heap, you can enter the following:

RDMP:*dumpfile$d* HEAP *[/address] [/format]*

If the data you wish to display is not in the Control Heap, you must instead retrieve the data from a bank identified by a bdi. (See Requesting data from a specific Extended Mode bank.)

Be aware that all the HEAP requests are used to view the raw heaps. For symbolic viewing of variables, use the STACK functions.

Extended Mode ALS requests

The ALS (Activity Local Stack) provides all programs and their internal subroutines a dynamic memory area where they can save registers or interface call packets when needed. In general, the ALS is a stack where programs keep data that can be discarded when the current invocation of the program terminates. For the following request, The Visual Dump Analyzer will traverse the ALS stack and determine the size and location of all ALS frames that make up the stack:

RDMP:*dumpfile$d* ALS

In addition, ownership of each of the individual ALS frames is shown by displaying the name of the internal subroutine that allocated it. To do this, The Visual Dump Analyzer analyzes the AFA (Activity Fixed Area) within the ALS frames and translates each entry to the name of the subroutine and the line number occupied by this subroutine within the program. This provides useful knowledge of where the ALS frames are active in the program.

Because UCS Fortran allocates an ALS frame for all subroutines – internal and external – you can use this request to show the path taken though the program from the very beginning of execution until the time of the abort. This easy and powerful method gives a thorough overview of where the program was at the time of the abort along with the complete path it took during execution.

With contingency aborts, you can also use the ALS request to determine whether the contingency occurred within the Basic or Extended Mode. If the first ALS frame is the NPERTS frame, you can conclude that the program aborted while in Basic Mode. That is, RTS uses this frame as a save area for Extended Mode while executing Basic Mode code. It should be noted, this assumption can only be made with contingency aborts because the XTCA abort code is itself Basic Mode code and XTCAs will therefore always terminate from within Basic Mode.

To view the ALS frames for a specific program including any external subroutines that are linked to it, you can use the following request:

RDMP:*dumpfile$d* ALS *[/program] [/format]*

Finally, to view the raw ALS data from a given address within the Control Heap, you can use the following request:

RDMP:*dumpfile$d* ALS *[/address] [/format]*

Extended Mode RCS requests

The RCS (Return Control Stack) is shown frame by frame with The Visual Dump Analyzer translating all frame data into understandable text. This is accomplished by merging data from the HVTIP table in the PCT with data from the RCS. Combining these two tables enables The Visual Dump Analyzer to determine the actual program from where an onward call is made. Once this is done, the Virtual Address of the RCS call can be translated into a source name and a line number. To request this data, enter:

RDMP:*dumpfile$d* RCS *[/RAW] [/format]*

The RCS is maintained as a stack with calls and returns within UCS programs *and* HVTIP banks resulting in either an entry being pushed (on calls) or popped (on returns) on this stack. This logic resembles that of the ALS, although RCS gives complete information on *all* calls made by the transaction, whereas ALS only provides information with Extended Mode programs that allocate memory.

Because the RCS is an Exec bank that cannot be corrupted by the application program and because it contains a comprehensive recording of a transaction's execution path within UCS programs and HVTIP banks, the RCS is a tremendously useful tool in analyzing the state of an Extended Mode transaction when it aborts.

Rescheduling the Input Message

The Visual Dump Analyzer allows you to reschedule the input message that exists within a dump. There are many benefits with rescheduling input messages, all of which assist 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 – although 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 ***The Visual Trace – User Guide***).

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 the input message again for a normal VDU transaction, simply enter the following request:

RDMP:*dumpfile$d* RESCH

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.

RDMP:*dumpfile$d* RESCH *[/pid] [/valtab]*

This will reschedule the input message – with the requested PID and VALTAB overriding the default if they are entered. Finally, if you have set traces on the PID used for rescheduling, you can now use TST:PRINT to capture the trace. Naturally, if the reschedule causes a new abort, you can open the newly created 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 desired PID number in the input:

RDMP:*dumpfile$d* RESCH / *pid*

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 use 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.

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 adding the EDI keyword to the input, you can request The Visual Dump Analyzer to remove the IATA HTH header and precede the input message with the "**EDIT**:" transaction code.

RDMP:*dumpfile$d* RESCH / EDI [/ *pid*]

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. In this way the transaction will be scheduled and the Edifact message will be routed directly to the Edifact Handler due to the "**EDIT**:" transaction code.

ProView Generated Programs

Programmed Views – or more commonly called ProViews – is a USAS High-level Automated Programming Language that is designed specifically for the USAS environment. By providing a high-level syntax, ProViews has been shown to reduce development time for many USAS programs to 10-20% of that used with conventional programming, with the resulting lines of code being reduced by a similar magnitude. Efforts to make future changes to a program's functionality are also greatly reduced by using the ProViews language. These efficiencies are primarily a result of automating the data access, data conversion and user-interface logic – with the resulting output providing a modern and intuitive interface for the end-user unlike those often seen in the USAS environment. (See ***ProViews - A USAS High-level Automated Programming Language*** by Gazelle Inc.)

As ProViews enables application programmers to develop application programs more quickly and with much less manually written code than traditional programming, more and more USAS programs are today written using this High-level Automated Programming Language.

ProViews are stored within a standard mainframe element called a View Element. This element is passed to the ProView pre-compiler, which generates a USAS Fortran program called a View Program. The program, in turn, is compiled and mapped into an HVTIP bank by using standard USAS procedures.

Traditional Source → Compiler → App. Program

ProView View Element → ProView Pre-compiler → Generated Source → Compiler → App. Program

When a ProViews generated program aborts, The Visual Dump Analyzer will analyze this it just like any other program. Because the actual program aborting is generated by the ProView pre-compiler (Generated Source) from the ProView View Element, The Visual Dump Analyzer will recognize the Generated Source as the true source code and calculate all symbolic information and line number references back to this source.

However, this is far from ideal while debugging these programs as you naturally want all symbolic references and line numbers to correlate back to the human written source (The Proview View Element).

This is a common problem to all High-level Programming Languages, however, ProViews has recognized the problems and has overcome it by leaving helpful debugging information in the Generated Source code so that The Visual Dump Analyzer can correlate all symbolic information and line number references back to the initial View Element.

In most cases, The Visual Dump Analyzer will recognize the ProView Generated Sources and will automatically correlate back to the ProView View Element. However, you can also force this by informing The Visual Dump Analyzer that ProViews has generated the source by adding the PROV keyword to the dump analysis:

RDMP:*dumpfile$d* PROV [/ *program*]

This will correlate all line number references back to the human written ProView View Element instead of the automatically Generated Source for the aborting program or for the program specified in the input.

The Visual Dump Analyzer's Knowledge Database

The Knowledge Database allows you to save hints and help information related to a specific abort code and program. These hints, which can contain *expressions* resulting in detailed explanations, are automatically displayed the next time an abort occurs that has the identical conditions.

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 considered to be small *views* for their constructs are the same: freeform text and *expressions*. To request a display for logging information to the Knowledge Database, enter:

RDMP:HINT

Hints are associated with dumps by their program name and XTCA error code. Because they are valid for all programs, unique XTCA errors produced by *ACB routines* should use **ALL** as the program name. This will result in the hints being displayed whenever its related error code occurs throughout the system.

The Visual Dump Analyzer Knowledge Database contains three types of information: the **Reason for abort**, **Hint text** and **X11 Hint**. To display or update the hints associated with a particular program, enter:

RDMP:HINT / *program* / *xtca-error-code*

And to display or update a hint associated with an ACB routine, enter:

RDMP:HINT / ALL / *xtca-error-code*

If you wish to update a hint within the database, just enter the program name and XTCA error code, and the hint will be shown in the update mask. After you have added or changed the information within the view, transmit to update the hint.

To delete a hint, first retrieve it and then delete all text from the **Reason for abort**, **Hint text** and **X11 Hint** sections. Next, transmit and a message will be sent confirming the deletion of the text.

For sites using both the Basic Mode of The Visual Dump Analyzer (RDMP:) and The Visual Dump Analyzer Client on a PC, the Client should be used to update information in the Knowledge Database. This is because the Client will automatically update *all* systems known to The Visual Dump Analyzer whereas the Basic Mode will only update the system where the update is performed.

Reason for abort text

The **Reason for abort text** is a general explanation of the abort that is often fixed or factual in nature. Being composed of freeform text, it resides in a separate record of the database from the **Hint text**. If the **Reason for abort text** 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 an ability to obtain the source code associated with this abort, if needed. Furthermore, it is also helpful when you are later viewing this hint.

Hint text

The **Hint text** is composed of freeform text and *expressions*. It is often dynamic in nature with changes to it being encouraged whenever a programmer makes discoveries while solving a dump that should be documented for future reference. By using *expressions*, you can create very informative hints by incorporating values from the dump so they will be immediately available 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 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 reuse the X11 (return) register. When these masm subroutines abort, it can be difficult to find out where these subroutines were 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*

An example of a helpful **X11 Hint** is with the substring routine F2SUBSTR, which 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, you would likely have to spend considerable time studying the system software before you would recognize that IBNOTE(1) actually holds the X11 return address. Then, you would need to convert the absolute address 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 can always be solved accurately.

Reading Absolutes and Zooms (RABS)

Request for decoded masm

The Visual Dump Analyzer can read absolute elements and display all the debug information stored within them by using the RABS: transaction and a variety of requests. The displays generated from these requests are all formatted according to the nature of the data – with no need for a user to interpret raw or unedited data.

All RABS: requests require an ***absolute-element*** as the first parameter, with it having a typical format as follows:

RABS: [qualifier*] [filename.] absolute-elt / version

Both ***qualifier*** and ***filename*** are optional fields. If omitted, the ***qualifier*** is the same as used for the USAS applications and, if omitted, the ***filename*** uses The Visual Dump Analyzer's *system* and *personal* search path.

When a *batch program* causes a dump, the aborting absolute element is automatically written to the dump file. Therefore, if you wish to analyze the contents of this aborting absolute, you can supply the ***dumpfile$d*** file name as the input parameter to this processor and you will be allowed to view it in the same manner.

RABS: dumpfile$d

In this case, the ***qualifier*filename.absolute-elt / version*** parameter is replaced with information taken from the ***dumpfile$d.***

Request for decoded masm by address

To view the I-Bank of an absolute element, you can request The Visual Dump Analyzer to decode the I-Bank and present the information in a format specifying the absolute and relative addresses along with relocatable names and names of the internal subroutines. The general request is as follows:

RABS:*absolute-elt [start-point]*

The display will include the next 250 instructions, even if this means a change of relocatable or other logical boundary within the absolute.

Along with the actual masm code, whenever possible a Fortran-like syntax is added to the display. Besides the source line numbers and labels, this includes supplying symbolic names for variable references, go-to statements and call statements. This extended syntax is especially helpful for non-masm programmers when they wish to follow the masm code and compare it to their source.

To quickly view the first masm instructions for the main program, just omit the start-point parameter as follows:

RABS:*absolute-elt*

Otherwise, you may choose to display masm instructions in the main I-bank at a particular ***absolute address*** by supplying it in the request using the following format (with a leading zero indicating an octal address):

RABS:*absolute-elt address*

In batch programs where each ACB is mapped in as its own bank, you may wish to specify the ***bank-name*** to display instructions within a particular bank:

RABS:*absolute-elt bank-name / address*

Or specify the related ***bdi-number*** of a bank to display masm instructions:

RABS:*absolute-elt bdi-number / address*

Sometimes it may be easier to just specify the ***entry point*** where you wish the display to begin and let The Visual Dump Analyzer determine the bdi value for the bank. This is especially useful when specifying the *start-point* in USAS ACBs.

RABS:*absolute-elt entry-point*

Request for decoded masm by FTN line number

You can also use Fortran line numbers to specify the start-point of a display by entering the (decimal) line number. The following request assumes the line number is within the main program:

RABS:*absolute-elt ftn-line-number*

If you wish to redirect the start-point to be in a particular ***relocatable***, just supply the relocatable name within the request. The following will cause the display to start at the *beginning* of the specified relocatable:

RABS:*absolute-elt relocatable*

Further, if you wish to refine the output to a particular ***line number*** within a relocatable, just add this to the request:

RABS:*absolute-elt relocatable / ftn-line-number*

You can also specify a relocatable ***relative address*** (leading zero indicates an octal address) following the relocatable program name:

RABS:*absolute-elt relocatable / relative-address*

And finally, you may specify the name of an internal Fortran ***subroutine*** as the start-point in this way:

RABS:*absolute-elt relocatable / subroutine-name*

Help Requests

You can request online help to assist you in formatting your requests for The Visual Dump Analyzer by entering the following:

RABS:*absolute-elt* HELP

This will generate a list of all the possible inputs for this absolute element with the name of the absolute element inserted in each input line along with tab stops allowing you to quickly move to the input you are interested in.

To view a more general help file, use the above HELP request to display *The Visual Dump Analyzer Basic Mode Command Sheet*.

RABS:HELP

Request for Absolute Debug Tables Displays

Absolute elements contain several tables used for both debugging and loading of the absolute. Each of these tables is displayed in an edited format with no need for interpreting raw data.

The MAP list request

The **MAP** request gathers information from a variety of the absolute's element tables and merges this information into a display that describes the mapping of the absolute.

RABS:*absolute-elt* MAP

This display provides all the information normally found in a map listing from the collector, thus eliminating the need to save the map listings.

The DIAG Table (Variables)

The **DIAG** table contains information from the symbolic dictionary entries (variables) of the absolute element. Because this is the table used to generate the STACK display during dump analysis, the **DIAG** and STACK displays are similar in format with one entry (variable) per line. Each line contains the variable type, stack relative address and absolute address.

RABS:*absolute-elt* DIAG

The output contains all the external subroutines with their variables being listed in alphabetized order. Internal subroutines are included within each external subroutine section, with their variables also being alphabetized.

The Segment Name (SNT) Table

The Segment Name Table (**SNT**) contains the names of all the segments within an absolute element.

RABS:*absolute-elt* SNT

Typically, USAS programs only contain one segment called **$MAIN$.**

The Element Name (ENT) Table

The Element Name Table (**ENT**) contains the names of all relocatable elements included in the mapping of an absolute element along with the date/time stamp when each one was compiled.

RABS:*absolute-elt* ENT

This table can be especially useful when determining the version (i.e., the date and time of compilation) of each relocatable within an absolute element.

The Bank Name (BNT) Table

The Bank Name Table (**BNT**) contains the names and BDI values of all banks within an absolute element.

RABS:*absolute-elt* BANKS

Use the **BANKS** request to get an overview of all the banks in the absolute element along with their BDI values, types and sizes.

The Entry Point (EPNT) Table

All referenced entry points within an absolute element are contained in the Entry Point Table (**EPNT**). Entry points that exist in an absolute element but are not actually *called* will not be included in this table.

RABS:*absolute-elt* EPNT

All entries within this table can be used as *start-points* in the decoded masm displays. The display of the **EPNT** table includes the absolute address for each entry point, the relocatable name of the elements that hold the entry point and the relative address within the relocatable.

The Absolute Value (ABSV) Table

An absolute element has one miscellaneous table: the Absolute Value Table (**ABSV**). It is a conversion table for all absolute names used by the collector whenever it needs absolute values. For example, it contains the BDI-values for the referenced bank names and external entry points that are called by an absolute element.

RABS:*absolute-elt* ABSV

This display provides an alphabetized list of the Absolute Value names and their corresponding octal values.

Request for ZOOM Symbolic Debugging Dictionary (SDD) Displays

Zoom elements contain a vast amount of debugging information in the Symbolic Debugging Dictionary (SDD). This information is gathered and represented in an easy to understand way by The Visual Dump Analyzer. Be aware that these RABS Extended Mode requests will handle and accept all kind of OMs, BOMs and ZOOMs as the input absolute.

The LINK list

The LINK request gathers information from a variety of the zoom element's SDD tables and merges this information into a display that informs of the general linking of the zoom. Bank Groups, the individual Banks, and the Bank Parts that may be grouped into a single bank are included within the output of this request:

RABS:*om-elt* LINK

The display will provide various information describing the elements used to construct the OM including name, date/time of creation, size, type and bdi value. Because the information on OM internal banking structure is also given, the display provides all the information normally found in an output listing from the linker, thus eliminating the need to save these listings.

Because the **LINK** request can sometimes be more extensive than is necessary, The Visual Dump Analyzer can limit the output to only provide data on the actual Banks and Bank Groups - thus suppressing the Bank Parts. To limit the output, simply enter:

RABS:*om-elt* BANKS

The SDD list (Variables)

Among other things, the **SDD** tables contain information on the symbolic dictionary entries (variables) of the OM. Because this is the table used to generate the STACK display during dump analysis, the **SDD** and STACK displays are similar in format with one entry (variable) per line. Each line contains the variable's basic type, name, addressing type and structural information. To request this, enter:

RABS:*om-elt* SDD

The output contains all the internal and external subroutines with their associated variables being listed directly after. You can use this listing to determine which variables exist in an OM, and which internal subroutine actually declares them. You should not use this request to find the address of a variable in a dump, however, as the great variety of variable access methods used by UCS programs along with runtime relocation makes this an almost impossible task. Instead, you should use the STACK request, which will give you the exact location of a variable in the dump.

The Element Table (ET)

The following request gives you information on where the OM elements used to LINK the zoom were found – including the specific files and the date/time of its creation:

RABS:*om-elt* ET

Actually, this request provides only one piece of information that is not shown in the LINK request: the actual file name from where the OMs were included. Because this can often be critical data while performing analysis, this request has been supplied for this purpose.

Reading Source elements or SDF files (RSRC)

As an alternative to opening a demand session, The Visual Dump Analyzer allows you to read source elements and SDF files while remaining in HVTIP. This is especially helpful when viewing trace files or quickly inspecting source elements.

Reading source elements

By using the RSRC: transaction and supplying the element name as the first parameter, The Visual Dump Analyzer will place output into the paging file. The format for reading ***source elements*** is as follows:

RSRC: ***[qualifier*] [filename.] element [/version] [line-number]***

Both ***qualifier*** and ***filename*** are optional fields. If omitted, the ***qualifier*** is the same as used for the USAS applications and, if omitted, the ***filename*** uses The Visual Dump Analyzer's *system* and *personal* search path.

The **line-number** parameter refers to the actual line number within the source element, and ignores CTS line numbers when they exist within the element. If the **line-number** is not included in the input, the display will begin with the first line of the source element. Specifying a line number is especially useful when you know the lines you wish to view or when the output exceeds the capacity of the paging file.

By combining the RSRC: function for displaying the contents of a file or element and the RPAG: function for traversing within the output and searching for particular text, you can efficiently view this information without having to open a demand session.

Source elements with CTS line numbers

When one or more CTS line numbers exist within a source element that is included within an absolute, all line numbers related to that compiled program that are reported by RDMP: and RABS: will be CTS line numbers. Further, when a source element without any CTS lines numbers is changed by a CCF that contains CTS line numbers, the combined source will be based upon CTS line numbers at the point of the CCF correction and beyond. Thus, in this case, *all* of the absolute's diagnostic tables will be based upon CTS line numbers.

To request the source element to be displayed with CTS line numbers whenever they are present, just enter the CTS keyword as follows:

RSRC: ***[qualifier*] [filename.] element [/version]*** **CTS**

To distinguish between normal and CTS lines within a source element, CTS lines numbers are followed by a ":" (colon) and normal line numbers are followed by a "." (period).

You can request the source element to be displayed starting at a specific CTS line number by using the following format:

RSRC: ***[qualifier*] [filename.] element [/version]*** **CTS** / ***cts-line-number***

If you experience a problem in having the line numbers from RDMP: or RABS: match your source element, check for the existence of CTS line numbers. Unfortunately, an absolute's diagnostic tables do not indicate whether they are based on normal line numbers or CTS line numbers. Therefore, RDMP: and RABS: are not able to warn you that calculated line numbers are actually CTS line numbers.

Reading program file TOCs
The format for reading a program file's table of contents (TOC) is as follows:

RSRC: TOC / ***[qualifier*] program-file.***

This will produce a list of all the elements in the program file with deleted elements including the text **DELETED** at the end of the line.

Again, If omitted, the ***qualifier*** will be the default qualifier for the TIP system, typically **TIP$**.

Reading SDF files
The format for reading ***SDF files*** is as follows:

RSRC: ***[qualifier*] sdf-file. [line-number]***

If omitted, the ***qualifier*** will be the default qualifier for the TIP system, typically **TIP$**.

The SDF-file can be a symbiont file or any SDF format as long as the length of its lines does not exceed 132 characters. The SDF lines can be either fieldata or ASCII character lines.

Reading records from SDF data files
The format is the same as for reading ***SDF files*** but includes the **REC** keyword as follows:

RSRC: REC / ***[qualifier*] sdf-file. record-number***

If omitted, the ***qualifier*** will be the default qualifier for the TIP system, typically **TIP$**.

The records may be up to 504 characters long with the contents of each record being broken into separate lines to ensure the entire record is visible. Further, each record is preceded with a brief header informing on the record number, and column settings indicating each 10th character for ease of calculating data positions within the record. The records can have either a fieldata or ASCII format.

Help Requests
You can request online help to assist you in formatting your requests for viewing SDF files or source elements with The Visual Dump Analyzer by entering the following:

RSRC:*[sdf-file or element]* **HELP**

This will display general input formats for the RSRC: functionality.

VDA enhanced paging (RPAG)

The Visual Dump Analyzer includes an enhanced paging feature. This feature, which is built on top of standard USAS paging, makes working with large USAS paging files easier and more efficient.

The enhanced paging feature is designed to work with all The Visual Dump Analyzer requests. Being able to be set/unset within your personal setup (refer to RDMP:SETUP, `DISPLAY PAGING LINE AS LAST LINE` parameter), when this feature is set it will be automatically activated with every request for The Visual Dump Analyzer. Note that normal USAS paging also functions when this feature is turned on.

Paging functions

The basic functionality of the enhanced paging is to add a *lazy line* as the last line of the screen's paging area – typically the last line on the screen. The *lazy line* contains quick, preset inputs for The Visual Dump Analyzer to page your output up and down.

>RPAG:PN >RPAG:PB >RPAG:PF >RPAG:PL >RPAG:PNH >RPAG:PBH >RPAG:SC >RPAG:

Besides the normal Page Next (**PN**), Page Back (**PB**), Page First (**PF**), and Page Last (**PL**) options, you may also select the Page Next Half (**PNH**) and Page Back Half (**PBH**) options. Further, you can also specify a specific number of lines to be paged up or down:

RPAG:+15 or **RPAG:-15**

Use these inputs to align your pages exactly as you would like without the constraints of standard USAS paging. Entering one of the standard USAS paging functions will automatically turn off the enhanced paging feature.

The search functions

Enhanced paging also allows you to **search** paging files by using the **SC** sub-function:

RPAG:SC

Entering this results in a second line being displayed above the normal *lazy line* allowing you to enter the search string:

>RPAG:SEARCH/<SEARCH TEXT>
>RPAG:PN >RPAG:PB >RPAG:PF >RPAG:PL >RPAG:PNH >RPAG:PBH >RPAG:SC >RPAG:

Replacing **<SEARCH TEXT>** with a string of text to search upon (and transmitting) will initiate a search beginning at the first line of the visible page. The first matching text will be moved into view with the located text being displayed in *reverse-video*.

To find the next occurrence, simply transmit again and the search will continue in the same manner. When the search text is no longer located, the response "**- NOT FOUND**" will be placed at the end of the input line.

Switching between Paging Levels

The enhanced paging feature allows you to view all five paging levels in normal USAS paging.

To activate enhanced paging for a ***function-code*** that has data stored in the paging file, but has not automatically initiated this paging feature, you can activate it manually by entering:

RPAG:*function-code*

Save Requests

You can save the contents of the paging file in an element of a program file by using the SAVE function:

RPAG:SAVE / ***file.element***

Once the paging file is stored as an element in a program file, you can perform all the normal actions such as reading, editing and printing upon it.

Help Requests

You can receive an overview of the function codes that currently have data in each of the five USAS paging levels by entering:

RPAG: or **RPAG:HELP**

The function codes will be shown at the top of the display with formatted inputs and tab stops allowing you to easily switch between the active USAS paging levels. This output also contains a quick help file describing this enhanced paging functionality.

Displaying Aborts (DABT)

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 two main purposes. First, it can provide the user with a quick *System Abort Status* by presenting an overview of all aborts that have occurred on the system during the past specified period. This includes a list of all the aborts along with diagnostic information and preset inputs for easily retrieving the dumps. Second, 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.

Selecting Aborts

Commonly, a programmer is interested in viewing all the aborts that have recently occurred on a system. With the most recent aborts being displayed first, this can be done by entering the following:

DABT:*[date]*

The paged display of this *System Abort Status* request will list the date and time of each abort, the aborting program name and function code, and some basic diagnostic information. For XTCA aborts, the diagnostics include the decimal ibserr/ibdiag values, and for contingency aborts it includes the contingency code and abort address.

The optional ***date*** parameter can be used to limit the output to aborts occurring in a seven (7) day period beginning on the specified *date*. When omitted, today's date is used as the default with the aborts occurring in the past one week being listed. Finally, the **ALL** parameter can be used in place of a date to view all aborts that are stored in the SED$Q common bank.

This *System Abort Status* display can also be filtered by a specific application, function code or program name as follows:

DABT:*application [/date]*
DABT:*function-code [/date]*
DABT:*program [/date]*

When ***application*** is specified, all aborts that have a function code belonging to the specified application along with all aborts occurring in programs belonging to the specified application are selected. That is, aborts that have either started or ended in the specified application will be listed.

When ***function-code*** or ***program*** is specified, only aborts with the specified function-code or aborts having occurred in the specified program are displayed.

Occasionally, it may be interesting to only view aborts that have occurred in batch programs. This can be requested with the following request:

DABT:BATCH *[/date]*

Finally, the *System Abort Status* display can be filtered by specific diagnostic information by entering either of the following:

DABT:*ibserr* or *ibnote* or *ibdiag* *[/date]*
DABT:*contingency-code [/date]*

When a decimal value (without a leading zero) is supplied as the above parameter, all aborts having a matching ***ibserr, ibnote*** or ***ibdiag*** value will be displayed. Similarly, when an octal value (with a leading zero) is supplied, all aborts having a matching ***contingency-code*** value will be displayed

Check Marking Aborts

The Visual Dump Analyzer allows you to set an indicator flag – or check mark – on an abort. This check mark is often used to indicate that a particular abort has been selected by a programmer for study, thus avoiding other programmers from duplicating this effort. Check marks can be set for a particular abort by first entering the function:

DABT:CHECK *[/date]*

This request will then produce a list of aborts that includes a column of preset functions that can be used to toggle the check mark indicator related to a dump. By transmitting on the input string in this column, the abort's check mark value will be toggled. The column to toggle these values contains the following functions:

DABT:dumpfile$d(1)

Finally, when a check mark has been set for a particular abort, it will be noted with an asterisk ("*") in the **CK** column of the DABT: display.

Controlling Capture Mode

When problems occur on the system that result in many aborts in a short period of time, an excessive amount of resources will need be used to create the actual dump file related to each abort – often adversely affecting the system's overall response times. The Visual Dump Analyzer can assist with this situation by limiting the number of aborts that are actually processed by the USAS SYS contingency handler. In these circumstances, the environment can be setup to only process unique aborts and to limit the overall number of aborts that are processed each minute.

The Visual Dump Analyzer can capture dump files in three different modes: ALL, SELECTIVE and AUTO. When ALL mode is set, all dumps will be processed up to the overall dump limit that has been configured within a one minute period (see Controlling Dump Limits section). Further, when SELECTIVE or AUTO modes are set, only new aborts (i.e., those having unique diagnostic information) will be processed in a one minute interval, with a counter being incremented for aborts that have already been captured in the past minute.

The ALL and SELECTIVE modes can be manually set by using the following functions:

DABT:SAVEALL … and
DABT:SELECTIVE

The AUTO mode cannot be manually set, but is automatically set by the system when the *auto-limit* (see Controlling Dump Limits) is reached within one minute. Reaching this value, which is the number of aborts that are identical in nature to others within one minute, indicates significant problems may exist on the system and it is in need of protection. When one minute passes without any dumps on the system, it is assumed to have stabilized and the ALL mode is reestablished.

When in SELECTIVE or AUTO mode and an abort is not captured because it is identical to a dump already taken, a DUMP ALREADY TAKEN message will be sent to the bottom of the screen in place of specifying the dump$d file name used to contain the dump.

Some dumps will always be processed by the system regardless of whether any of these dump limits have been reached. These include BATCH aborts and XTCA aborts having an ibserr value of 591 as these are both considered to be requested by the user.

The current mode along with the configured dump limits are shown in the header line of the DABT: display.

Controlling Dumps Limits

The dump limits that control the way aborts are processed on the system can be configured to the values that are determined to be optimum for an individual system. These values are set by using the following function:

DABT:SETLIMITS / max-dumps / max-status-msg-prints / auto-limit

The first value, ***max-dumps***, is the maximum number of aborts that are processed during a one minute period.

The second value, ***max-status-msg-prints***, is the maximum number of DUMP ALREADY TAKEN messages that will be directed to the status printer during each minute (if status printers are configured at the site).

The third value, **auto-limit**, is the number of aborts that are identical to others that have been processed during a one minute period while in ALL mode (i.e., number of aborts that would *not* have been processed if in SELECTIVE mode). When this limit is reached, the AUTO mode will be established.

If any of these limits is set to zero, that limit is inactivated and an unlimited number of dumps can be processed within a one minute period.

The default dump limit values are 10/10/20. To set these default dump limit values, simply enter the following:

DABT:RESET

Controlling The SED$Q common bank

A few additional functions exist for controlling the SED$Q common bank itself. To clear the SED$Q common bank of all its recorded abort entries, use the following input:

DABT:CLEARIT

The SED$Q common bank is actually checkpointed to an SDF file named USAS*SED$Q every minute. This occurs with the first abort or first **DABT:** display request within the minute. Then, when the system boots up, the SED$Q common block is automatically recovered from this checkpoint file. Furthermore, if the need arises, this checkpoint file can be copied into the common bank using the following request:

DABT:RELOAD

Help Requests

You can request online help to assist you in formatting your requests for displaying aborts with The Visual Dump Analyzer by entering the following:

DABT:HELP

This will display general input formats for the **DABT:** functionality.

Displaying Files (RFIL)

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 Status Report that informs how full each file is for all the FCSS freespace files on your system. Next, if a particular file stands out by being nearly full or the like, you can receive a more detailed analysis of it.

To produce a Quick Status Report, just enter:

RFIL:

This Quick Status Report will display one line for each FCSS freespace file with data being aligned in columns. This includes 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 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 report.

Selecting Files

Very often a programmer is only interested in viewing the FCSS freespace files belonging to a specific USAS application rather than all files on a system. Thus, the Quick Status Report can be filtered to only include FCSS freespace files belonging to a specific application.

RFIL:*[application]*

Below is an example of the Quick Status Report for the RES application.

```
RFIL:RES
.............................. FILE ANALYSIS .......................
NAME   NUMBER  REC-SZ  RTYP  D-VALUE     MAX RECS   AVAIL. RECS (%)
01FILE    333     112     8  4,75,5        950272        562230  59%
35FILE    335     112     8  5,25,6        196608        157556  80%
10FILE    343     112     8  7,55,8        569344        514048  90%
05FILE    344     112     8  7,45,8       2736128       1290342  47%
BOFILE    348     112     8  8,100,8       100352         92155  91%
FIFILE    349     224     1  1,100,1        12288          7443  60%
HIFILE    350     112     2  2,100,2      9134080       7559950  82%
59FILE    359     112     8  8,100,8        16384          8209  50%
IVFILE    361     112     8  8,100,8      2654208       1092432  41%
PNFILE    375     112     1  1,100,1      5144576       2161975  42%
PRFILE    377     112     8  4,75,5       8929280       4818508  53%
PXFILE    379     112     7  6,55,7       3153920       1791807  56%
06FILE    423     112     8  6,50,7        712704        412493  57%
ACCEPTED
```

Detailed File Analysis

The Detailed File Analysis Report produced by The Visual Dump Analyzer contains all the information needed by both system programmers that are responsible for creating and maintaining the files and application programmers alike. The detailed analysis presents the file information in a few sections with each one focusing on a specific aspect of FCSS freespace file usage.

You can request the Detailed File Analysis Report for a file that you would like to analyze by entering either its TIP file number or TIP file name.

RFIL:*[Tip-file-no]*
RFIL:*[file-name]*

This will produce a Detailed File Analysis Report that includes multiple sections. The initial section contains an overview of the file characteristics along with subfile information stating what subfiles are part of this TIP FCSS freespace file. An example is displayed below.

```
RFIL:333
.......................... FILE ANALYSIS .......................
NAME   NUMBER  REC-SZ  RTYP  D-VALUE     MAX RECS   AVAIL. RECS (%)
01FILE    333     112     8  4,75,5        950272        563420  59

01FILE SUBFILES
SUBFILE  1 - AF  - RES - FIRST CLOSE AVAIL FILE
SUBFILE  2 - AI  - RES - AVAILABILITY FLT. INDEX
SUBFILE  3 - CS  - RES - SCHEDULE CITY PAIR INDEX
SUBFILE  4 - FF  - RES - MASTER AND ADVANCE FLT
SUBFILE  5 - SR  - RES - SCHEDULE FILE
SUBFILE  6 - FS  - RES - FREE SALE AGREEMENT
SUBFILE  7 - SA  - RES - STATUS MESSAGE ADDR. CO
SUBFILE  8 - SM  - RES - MINIMUM TIME FILE
SUBFILE  9 - FC  - RES - FLIGHT NBR. TABLE RES
SUBFILE 10 - RS  - RES - SEATMAPS
```

The next section informs of the Free Bit Count (**FBC**), which represents the unallocated space for a file. Note that when an FCSS freespace file contains a D-value that is different from 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.

```
TOTAL BASIC RECS          TOTAL FBC (%)     FBC<D (%)     FBC>D (%)
FREE BASIC RECS              649892  68     533515  56     116377  12
RELEASE ONLY BASIC RECS       86259   9      65800   6      20459   2
ALLOCATABLE BASIC RECS       563633  59     467715  49      95918  10
```

As a consequence, the **FBC** is displayed both for the entire file and each logical file. As shown above, the data for the entire file is labeled **TOTAL FBC**, the portion below the D-value is labeled **FBC<D** and the portion above the D-Value is labeled **FBC>D**. 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 **FBC** is subtracted from the Free Basic Records to determine the actual records that are available for allocation by an application program.

When analyzing the real-time characteristics of a particular FCSS freespace file, one must also consider the space that is available for each basic record type. That is, studying the **FBC** figures as shown above is not enough. Therefore, The Visual Dump Analyzer also breaks down the freespace statistics by record type.

With all FCSS freespace files that have more than one configured record type, fragmentation of its allocatable space can become a problem. The D-value partially avoids this, however, it has not eliminated the problem. Therefore, knowing how fragmented a file is critical when determining a system's ability to handle new record allocation by the application programs.

RTYP	REC	WORDS	REC/FILE	REC/DVAL	(%)	REC/FREE	(%)	REC/FRAG	(%)
1	1	112	950272	718272	75	533515	74	0	0
2	2	224	475136	359136	75	257795	71	8962	3
3	3	336	316757	239424	75	160261	66	17577	9
4	4	448	237568	179568	75	116516	64	16862	12
5	6	672	158378	38512	25	17248	44	2144	11
6	10	1120	95027	23200	25	9711	41	1926	16
7	16	1792	59392	14384	25	4874	33	2396	32
8	32	3584	29696	6960	25	1632	23	1997	54

The **REC/FILE** column shows how many records are available for each type if the entire file could be used for this record type without considering the D-value setting. The next **REC/DVAL** column shows how many records are available for each record type with respect to the D-value. The **REC/DVAL** percentage informs on how much of the file is available to this record type due to the D-value setting. Continuing, the **REC/FREE** column informs on how many of these records are free and available for record allocation.

The **REC/FRAG** column calculates how many logical records are lost to fragmentation for each record type. Finally, the fragmentation percentage is calculated as how much of the empty space is lost due to fragmentation. The formula is a follows:

Fragmentation Percent = Basic Records lost to fragmentation*100 / Free Basic Records

A natural consequence of this formula is that when the file is filling up, its fragmentation percent will increase. This is actually a desired reaction as it provides a clear warning of when fragmentation is starting to become a problem.

Note that because fragmentation indicates records are lost to fragmentation for a particular record type, say record type 8, this does not mean the space is necessarily wasted in the file. The area may actually be useable by a smaller record, say record type 4, as long as the D-Value setting will allow for the allocation.

Further, a high fragmentation percentage does not necessarily mean that a problem exists as it also depends upon how the file is actually used, which is the purpose of the next section. It displays statistics on how many records of each record type have been allocated and released since a specified data – this date being stated in the **FILE GROWTH MESSURED FROM** line.

RTYP	REC	WORDS	#-ALLOC	(%)	#-FREED	(%)	GROWTH	(%)
1	1	112	180	26	95	19	85	0
2	2	224	480	69	375	76	105	0
3	3	336	10	1	2	0	8	0
4	4	448	4	0	4	0	0	0
5	6	672	6	0	6	1	0	0
6	10	1120	10	1	10	2	0	0
7	16	1792	0	0	0	0	0	0
8	32	3584	0	0	0	0	0	0

FILE GROWTH MEASURED FROM 11/09/08 08:00:01

A primary statistic within this display is the **GROWTH** column. When the statistics have been gathered over a period of time that can be considered the natural life cycle for the file, the Growth Percentage should be close to zero. If it is not, this is indication that the file is either being slowly filled up or slowly being emptied. Moreover, this can be an early warning that application program errors exist. If the file is filling up over a longer period of time, this often indicates that application program is creating orphan records.

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

I/O STATISTIC	FNC	CALLS	(%)
READ	RD	2717387	96
READ LOCK	RL	70635	2
LOCK	LK	0	0
WRITE UNLOCK	WU	20810	0
WRITE NO LOCK	WW	0	0
WRITE KEEP LOCK	WR	425	0
WRITE AND REALLOCATE	RA	0	0
ALLOCATE	AL	0	0
ALLOCATE AND WRITE	AW	690	0
ALLOCATE SPECIFIC	AS	0	0
RELEASE UNLOCK	RU	492	0

OPERATION	RATIO
READ TO WRITE	127.16
WRITE TO ALLOCATE	30.77
LOCK TO UPDATE	3.32

This display provides very useful information when, for example, designing data structures for the file or evaluating existing application programs that use the file. This information includes ratios, which describe how the file is logically used.

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

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

The **LOCK TO UPDATE** ratio informs on 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.

This statistical data is gathered over the same period as the **GROWTH** statistics.

The last section is produced from data generated by The Visual Dump Analyzer's own batch Freespace Scan, which scans through all the files on the system and captures data. If this batch program has never been performed, this section will automatically be excluded from the Detailed File Analysis Report. (See ***Freespace Scans***)

ALLOCATED LOGICAL RECORDS AS PER 11/08/08 20:00

SUBFILE	RT-1	RT-2	RT-3	RT-4	RT-5	RT-6	RT-7	RT-8
FF	435	179	115	193	1441	733	12	0
SR	567	497	428	439	856	11000	0	0
FS	2	0	0	0	0	0	0	0
SA	3	0	0	0	0	0	0	0
SM	0	0	0	0	0	0	0	1
FC	2	2	1	2	4	6	14	7
RS	25536	67010	1677	429	3	0	0	0

This section will inform on how many records of each record type are used by each subfile. Being the only place where this information exists on the system, it can only be obtained by running though the entire file and analyzing each allocated record. Because of the great processing that is required for this, it is not captured at request time, but instead during the last batch Freespace Scan.

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 guess that a D-value change will eliminate a fragmentation problem within a file, you can determine with certainty if this is true. 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 or what-if D-value, enter:

RFIL:*Tip-file-no/d-value*

The specified D-value should be entered as a normal D-value in the format **record-type,pct,record-type.**

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 PCTs 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. You would have to change the D-value to 47% to actually create an effect.

Detailed BITMAP Analysis

The Visual Dump Analyzer allows you to pictorially view an analysis of the actual contents of a BITMAP. Further, when a BITMAP is analyzed, its date is checked against the actual file to ensure they correspond with one another.

To request the BITMAP analysis, enter:

RFIL:*Tip-file-no* / BITMAP/ *bitmap-no*

This will make The Visual Dump Analyzer include a BITMAP section in the Detailed File Analysis Report showing the BITMAP as seen below.

```
BITMAP START-REC   END-REC      FBC (%)     FBC<D (%)     FBC>D (%) STATUS
     1      2049      4096     1471  71      1211  59       260  12 UP

.......................... RAW BITMAP FOR RECORDS<D ..........................
22222233311-4444-----------22--------1------1333333---22--221-----------22------
---122-------22-------------------------------------------------------2222------
--122--22-----2222-----2222---------------2222-------1--------22--1------------
-------22----1---------122--------1-----------1--22------------221--22--------22
---1-----22--------2222--22--22----1--------------1-----------22----------------
---1--22---------1---------22--2222-------------------22221----2222--------22---
-----22222222------2222---1-----1----------------------1----22-------22---------
----2222-----------------22--------------1----221-22-----555555---------1-22-22
------221--2222-------22-----22-----------22--22--22------------122-22---22--122
1--22----------22----------------------22--2222-2222----2222-----22-22-----22222
2-------------------22--------2222---122-22------------------------22---1------
---22---------2222--22----222222-----1----------1-------------------1------1---
-22-----------------22---------------------22--------------------22-----------
---22------------1----------122------1-----------22-1--------------22----------
---22---------------22-----------------2222---1------------------11------------
1-------1---2211-----22----2222-------------------------------------------22----
22-------------------------------------1--222222---------122----------22------
--------221----1-1--22-------------------22-------------22---22--------22------
---------------22----22-------------------------2222------------------------66
66666666-------------------

.......................... RAW BITMAP FOR RECORDS>D ..........................
6666666666-----------------------------------------6666666666----------66666666
66-----------------------------------6666666666----------6666666666666666666666
66666666--------66666666666666666666--------66666666666666666666------6666666666
6666666666------------6666666666--6666666666------66666666666666666666666666666
66666666666666666666----------6666666666----------6666666666-------------------
6666666666----------------6666666666-----------------------------------------
-------------------

1->8: RECORD IS USED FOR ALLOCATION OF RECORD TYPE 1->8
  - : RECORD IS FREE ACCORDING TO BITMAP AND FILE IS MARKED RELEASED TOO
  F : RECORD IS FREE ACCORDING TO BITMAP BUT FILE CONTAINS USAS CONTROL WORDS
  M : RECORD IS FREE ACCORDING TO BITMAP BUT FILE IS MISSING FREE BIT PATTERN
  A : RECORD IS ALLOCATED ACCORDING TO BITMAP BUT FILE CONTAINS FREE BIT PATTERN
  E : RECORD IS ALLOCATED ACCORDING TO BITMAP BUT IS MISSING USAS CONTROL WORDS
```

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 – which include 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 are not 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***)

Note that the BITMAP analysis can be combined with the What-IF D-value feature. This will then provide you with an accurate picture of the effect that a What-IF D-Value adjustment will have to a specific BITMAP.

RFIL:*Tip-file-no* / *d-value* / BITMAP/ *bitmap-no*

This logic should be used to verify the consequences of a D-Value adjustment before changing the actual D-Value using FREIPS.

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. The data will then be available for later use by both RFIL: and The Visual Dump Analyzer GUI (VDA.EXE).

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

Furthermore, this batch run will enable The Visual Dump Analyzer GUI (VDA.EXE) to produce the historical graphs that display the evolution of each subfile over a period of time.

The Freespace Scan can also be stated on demand by directly entering the above input. Generally, however, scanning all the files once a day should be sufficient.

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 with its use. (See ***The Visual Dump Analyzer – Scanning Freespace Files***)

This page has intentionally been left blank
(for printing purposes)

Table of Contents