z/XPF®

High Definition Profiling

Installation Guide

Duke Software www.duke-software.com

For Releases V2R2 and later (patent pending)

Revision date: 05-27-2014

PREFACE
PROPRIETARY LEGEND

Contact Information

Phone 281-395-5570
E-Mail: David.Day@duke-software.com

Please visit www.duke-software.com for the following services:

- General information about z/XPF
- E-mail links to Marketing and Customer service
- FTP links for uploading diagnostic information to Technical Support

Table of Contents

Welcome to the z/XPF Installation Guide

Thank you for your decision to install and use z/XPF. The installation process has been designed to be easy, and straightforward. However, if you should need our help, please contact :

David Day: 281-395-5570 - David.Day@duke-software.com

STOP. Please read this.

Everyone is busy, with a huge task list. We're all in a hurry to get to the next job. However, z/XPF is SO different from other profilers that we urge you NOT TO SKIP STEPS when installing it. For example, z/XPF's abilities to measure DB2 applications will not work if you haven't installed that portion of the product.

The installation itself is straightforward enough. Please follow all the steps.

z/XPF runs at a very low level in the z/OS environment. z/XPF reads Trace Records which are generated by the inter-action of every program executing on a z/OS image with the operating system. There can be millions of trace records generated for any application, which means that:

- z/XPF has to execute at high enough priority to examine Trace Records before the Trace Table wraps. The factory default logic will issue an operator command to set the server address space to the SYSSTC WLM workload class. In some environments where there are many address spaces already assigned to this class, this may not be enough priority to insure that z/XPF gets control often enough to keep abreast of the activity occurring in the LPAR. The factory default logic will also set the number of processor trace buffers to 512 from the z/OS default of 256. If/when the ZXPFLOG contains messages indicating a loss of time, or gap, in the copying of trace records, consider setting the number of processor trace buffers higher than 512. This can be accomplished via an operator command, or using a control statement in the z/XPF start-up control dataset.

- z/XPF profiles job steps, not entire jobs.

- If the target application you are profiling is extremely active, and you set z/XPF to run the entire span of the jobstep, then you WILL consume a fair amount of available DASD space for the VSAM capture dataset. It is impossible to predict the number of cylinders needed to contain the complete data capture. It is driven by the activity of the target application. Consider setting aside a separate pool of available space for capture da-

tasets if DASD is at a premium in your installation.

- You have to wait for the data capture to terminate to be able to produce statistics. The environmental data that allows z/XPF to make 'sense' of the trace data is written to the end of the dataset at data capture termination.

So, while other profilers are like field glasses, z/XPF is more like an electron microscope: very useful in taking VERY granular measurements at an extremely low level. That's why the 'factory default' setting for any data capture session is 500K records. We think that's enough data for you to make an informed judgment.

We're convinced that there's nothing more useful than z/XPF for getting definitive information on resource consumption at the instruction level, but you have to set up and use the tool properly. *A hammer makes a poor screwdriver.*

We'll gladly help you, so call us whenever you have a question. OK, let's move on.

What is z/XPF?

z/XPF is a utility program that can capture, and tabulate performance data on running programs. It can easily identify how a program uses (or abuses) system resources, and can help programmers streamline their code for maximum efficiency.

z/XPF's basic architecture is unique (patent pending). Instead of "freezing" the target address space in order to gather information, z/XPF "wakes up" periodically (50 times per second) and reads information directly from the system trace table.

The system trace table is a set of buffers created for each processor. Records are written for many events in the system including SVCs, start subchannel, interrupt processing and many other significant events during application processing. Trace buffers are filled quickly and when they are filled, the trace buffers will over-write at the beginning, which overlays earlier trace records. This is called "wrapping". If trace buffers wrap before z/XPF can gather the records, they will be lost.

Because z/XPF reads trace records out of the trace buffers, it is far more effective at capturing events than other such products. HOWEVER, z/XPF depends on two very important principles in order to do its job:

1. **z/XPF MUST execute at a very high priority.** (or it won't be dispatched often enough to collect its data).
2. **The system's processor trace table(s) must be large enough to minimize "wrapping" before z/XPF can gather trace records.**

System Requirements

z/XPF V2R2 will execute on z/OS Release 1.10 and above.

Duke Software sends z/XPF activation codes with the product. If, for some reason you don't have these codes, please contactDuke Software either by phone or email for a valid license code.

This is important enough to expand upon: *z/XPF needs to execute at the same level of priority as z/OS system tasks (often Service Class SYSSTC).* Otherwise, z/XPF will not be dispatched often enough to collect the trace records that the user requires. Therefore, during initialization, the z/XPF server task will execute an Operator command to reset its service class to SYSSTC. Certain portions of z/XPF's data capture process also operate as a Global SRB. Finally, z/XPF will issue the TRACE ST command to increase the size of the system trace table. These commands are detailed below in "z/XPF and Operator Commands"

There are two situations in which z/XPF's Operator command may not get the desired results:

1) If your installation blocks program-issued Operator commands, and
2) if your installation has changed its service class definitions in such a way that SYSSTC is not a priority class.

You may need to consult with your installation's personnel to resolve this potential conflict. Otherwise, z/XPF may not be able to do its job for you.

TSO/ISPF considerations

While future releases of z/XPF will give users the option of executing z/XPF's reporting in batch, the current release does its reporting entirely under TSO. Therefore some considerations must be made.

Assigning priority to TSO regions

z/XPF's report processing is broken into two phases:

Phase 1 distributes the events into the proper reporting accumulators.
Phase 2 creates the reports by processing the accumulators.

z/XPF is capable of capturing a surprising amount of raw data – *even millions of events!* Therefore, two factors influence the elapsed time it takes z/XPF to create its reports:

1) The number of events in the source capture dataset, and
2) z/XPF's access to the processor.

Most z/OS installations treat long-running TSO transactions somewhat poorly and will relegate them to the bottom of the CPU "food chain". They only get access to the processor when everything else in the system is satisfied. Therefore, the installation may wish to give these TSO sessions greater priority, or allow the creation of additional TSO userids for use in running z/XPF reports.

To illustrate this point, a customer recently sent us a data capture dataset containing just under 5,500,000 events. On his system, a 2096, it took almost a full hour to get through Phase 1. On our system, a 2097, it took just under 3 minutes. The customer was running on a heavily loaded system, while our own test environment is far less busy.

z/XPF and virtual storage

z/XPF trades virtual storage usage for speed. It is impossible to predict the amount of virtual storage that z/XPF will need to create reports because the amount of virtual storage required is dictated by the size of the dataset created during data capture.

With z/XPF V2R1 and above, both the z/XPF Server and the Report Generation functions use virtual storage obtained above the 2-gigabyte "bar". Therefore:

- **You MUST have enough space made available to your user's TSO userids to process the reports.** If your users get storage ABENDs during the report phase in z/XPF, that's an indicator that they need more space.
- **ISPF must be able to allocate storage above the bar.** Just how much storage needed above the bar depends on the number of Work Units and load modules that are reported on during z/XPF's data capture.
- **it is recommended that the installing Systems Programmer set the MEMLIMIT value in the SMFPRMxx member to NOLIMIT.**

z/XPF also expects to write to its ISPF log dataset. If no log dataset is allocated, then z/XPF will not be able to log ISPF error messages, and diagnosis of z/XPF problems within ISPF will be far more difficult.

Contents of the Installation package

z/XPF's installation files are made available via e-mail transmission from Duke Software. They can also be downloaded from the Duke Software's website. In either case, these files are compressed using WINZIP. However, in order to get the compressed file past certain e-mail filters on the customer's side, we rename the file extension from, ".zip" to, ".zxpf". Please rename the file extension back to, ".zip" and the decompression should proceed normally.
You will receive six files. They are:

1. README.txt
2. zxpf.VnRnMn.dmmddyy.initial1.txt
3. zxpf.VnRnMn.dmmddyy.initial2.txt
4. zxpf.VnRnMn.dmmddyy.initial3.txt
5. zxpf.VnRnMn.dmmddyy.initial4.txt
6. zxpf.VnRnMn.dmmddyy.xmit (one XMIT file)

The system files come named with multiple level qualifiers. The highest level qualifier is"zxpf", the next is a version-release-modification value, the next is a "date", the next qualifier identifies the type of the file.

Please review the accompanying README.txt file. It contains instructions on file allocations for the various install files.

Copying z/XPF files to the mainframe

Before starting this copy process, the installer must decide what high-level qualifier(s) will be used for all of the z/XPF datasets used during the install process on the z/OS system.

Consider using a two-level qualifier for z/XPF installations. The first qualifier would be "ZXPF" followed by a date in the format "DMMDDYY". If this approach is taken, then all datasets associated with a given version of z/XPF on your system will be easily identifiable.

Allocate four 1-track sequential datasets on the z/OS image. These datasets are JCL files you will tailor and submit to populate the z/XPF datasets necessary for the install process. z/XPF refers to these datasets as the INITIAL datasets.

zxpf.v2r2.initial1.txt is a skeleton JCL file. This job allocates all of the datasets used in the install process. You will run this job first.

zxpf.v2r2.initial2.txt is a job that will execute a TSO Receive to populate one z/XPF library. This library contains one member for each z/XPF library. This library is in TSO Transmit format. You will run this job second, after you have uploaded the XMIT file.

zxpf.v2r2.initial3.txt is another skeleton JCL file. This job copies each member of the XMIT library to individual XMIT datasets.

zxpf.v2r2.initial4.txt contains the JCL to execute a TSO Receive against the individual XMIT datasets created by the initial3 job. This job populates the z/XPF libraries that may be used as run-time libraries, or may be input to an SMP/E install of z/XPF.

Upload zxpf.VnRnMn.dmmddyy.INITIAL1.TXT to &HLQ.INITIAL1
Upload zxpf.VnRnMn.dmmddyy.INITIAL2.TXT to &HLQ.INITIAL2
Upload zxpf.VnRnMn.dmmddyy.INITIAL3.TXT to &HLQ.INITIAL3
Upload zxpf.VnRnMn.dmmddyy.INITIAL4.TXT to &HLQ.INITIAL4

The JCL contained in the &HLQ.INITIAL1 dataset will allocate all of the datasets needed for a non-SMPE install. Proceed to the z/OS platform, tailor the JCL to conform to your shop's standards, and run the INITIAL1 job.

NOTE: The PC file uploaded in step 2 was created on z/OS using TSO Transmit. It was then transferred to a PC using FTP in binary mode. This dataset, referred to as &HLQ.XMIT, will be used as input to the TSO Receive function. This datasest must be allocated as physical sequential, with an LRECL of 80, and a BLKSIZE of 3120. **It must be uploaded in binary format.**

After that job has been successfully executed, return to the PC to upload the binary file (which has a low-level qualifier of "xmit"). Upload zxpf.v2r2m'n'.Dmmddyy.XMIT to &HLQ.XMIT.

The directions above appear also in the README.TXT file that we send with the z/XPF distribution materials. HOWEVER, it's still a good idea to have a look at the README.TXT file as volatile changes will be documented there first.

NON-SMP/E installation

z/XPF may be installed without the services of SMP/E. Here are some instructions on how to do that. After these steps have been completed, please refer to “z/XPF and Installation Security” and follow the steps from that point.

Here is a summary of the steps you’ll need to complete for a non-SMP/E installation. They are covered in greater detail below.

1) Get the load modules into the proper libraries.
2) Get the JCL member used to start the z/XPF task into a proclib. Modify DSN’s within this member.
3) Get the clist used to invoke the z/XPF ISPF interface into a clist library. Modify DSN’s within this member.
4) Copy the ISPF panel, messages and table libraries.

That was the summary. Now for the details:

1) Get the load modules into the proper libraries.

If you choose to run the product from the &HLQ.INSTALL.LOAD library, then you will need to authorize this library. The &HLQ.INSTALL.LOAD data set should now contain the load modules needed to execute z/XPF. Within this PDS, you will find modules:

- APBEGN
- APDSNMDS (Does not need to be in an authorized library.)
- APD2STMT (Does not need to be in an authorized library.)
- APGETSQL
- APISPF (Does not need to be in an authorized library.)
- APLOGT
- APMAPP
- APMERGE
- APMRGECT
- APMVTTE
- APNTVCT
- APPBUFCP
- APPBUFCT
- APRESMGR
- APTBUFCP
- APTBUFCT
- APVSAM
- MERGCHNS
- PBUFCOPY
- TBFZOS10

The z/XPF server task must run authorized. If the decision is made to use the &HLQ.INSTALL.

LOAD library as the steplib when executing the started task, then you must authorize this library. If not, copy all the members on the previous page *except* APDSNMDS, APD2STMT and APISPF to an authorized library.

Members APISPF, APDSNMDS, and APD2STMT are the load modules used for the z/XPF ISPF interface. Either leave the members here in this library, or copy them to a load library that can be used by TSO/ISPF applications within your environment. The module COMPRES2 is a Program Object. Either leave COMPRES2 in the install library or copy it to a non-authorized PDSE.

2) Get the started task JCL into a proclib.

Within &HLQ.INSTALL.JCL is the skeleton JCL member needed to execute the z/XPF started task, ZXPFCPTR. Below is a copy of the jcl contained within this member.

```
//*
//ZXPFCPTR EXEC PGM=APBEGN,REGION=0M
//*
//STEPLIB  DD  DSN=&HLQ.APFLOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//INPUT    DD  DSN=&HLQ.CNTL(INPUTDS),DISP=SHR
//ZXPFLOG  DD  SYSOUT=*
//SYSABEND DD  SYSOUT=A
```

Copy the member to a proclib dataset as appropriate to your environment. Modify the dataset name on the STEPLIB DD statement to conform to the library you will be using to execute the z/XPF started task. Modify the dataset name on the INPUT DD statement to the library you will use for the input control statement(s) that the started task uses during initialization.

3) Get the clist moved to a clist library, and edit the clist.

The 2nd member of &HLQ.INSTALL.JCL you will need to move is ZXPFCLST. This is the CLIST used to allocate the z/XPF libraries, and invoke the ISPF interface. Below is a copy of the first few lines of code within the clist. You must modify these lines within the clist to specify the correct libraries. It is a good idea to shorten the name of the clist at this time from ZXPFCLST to ZXPF.

```
PROC 0 DEBUG LIST PRM('ZXPF') +
     LLIB('''&HLQ.APFLOAD'' +
     ''&HLQ.APFLOADZ''') +
     MLIB('''&HLQ.ISPMLIB''') +
     PLIB('''&HLQ.ISPPLIB''') +
     TLIB('''&HLQ.ISPTLIB''')
```

&HLQ.APFLOAD should be set to the load library that contains modules APISPF, APDSNMDS,

and APD2STMT. If you intend to run z/XPF from the install libraries, this would be set to &HLQ.INSTALL.LOAD, where the “&HLQ” value is set to the value you chose for this install.

&HLQ.APFLOADZ should be set to the load library that contains module COMPRES2. If you intend to run z/XPF from the install libraries, this would be set to &HLQ.INSTALL.ZLIB.LOAD, where the “&HLQ” value is set to the value you chose for this install.

&HLQ.ISPMLIB and &HLQ.ISPPLIB should be changed to the appropriate library names. See below.

Note that if the system name is changed from the default of ZXPF, the parm passed to the clist to at startup will need to be changed.

4) Copy the ISPF panel, messages and table libraries.

If you choose to run from the &HLQ.INSTALL.ISPPLIB, ISPMLIB and ISPTLIB libraries then this step is optional - you don’t have to copy anything. The ISPF panels needed are in dataset &HLQ.INSTALL.ISPPLIB and the messages are in &HLQ.INSTALL.ISPMLIB. &HLQ.ISPTLIB contains an ISPF commands table that allows the ISPF command REPEAT FIND to function when viewing z/XPF reports. You can either execute the ISPF clist using these libraries, or copy the members to an appropriate library for your environment.

SMP/E installation

If z/XPF is installed using the provided JCL, the &HLQ value used for the install may be the same value you used when creating the &HLQ.INSTALL and &HLQ.UPLOAD datasets. If installing into a common ISV SMP environment, you should be sure that the FMID, load module names, panel names, and messages are unique to z/XPF in that common environment.

At the end of the install, assuming you used a separate SMP CDS for the install, two load libraries will be created: &HLQ.APFLOAD and &HLQ.APFLOADZ. The panel library will be &HLQ.ISPPLIB, the messages library will be &HLQ.ISPMLIB and the table library will be &HLQ.ISPTLIB.

Skip steps 1 through 8 below if you are NOT installing z/XPF via SMP/E. We recommend you use a separate SMP/E environment for z/XPF.

1) In the install dataset run member DEFCSI to define the global, target, and DLIB CSIs for z/XPF.

2) In the install dataset run member DEFSMPE to define the SMP/E datasets needed for processing.

3) In the install dataset run member UPDTZNS to update the SMP/E global, target, and DLIB zones. We recommend that you use a separate SMP/E environment for z/XPF.

However, you may skip this step if you are installing z/XPF into a common program products SMP/E environment.

4) In the install dataset run member DEFTLIB to allocate target and distribution libraries used by SMP/E.

5) In the install dataset run member DDDEFS to add the DD definitions for the target and distribution libraries you allocated in the previous step.

6) In the install dataset run member RECEIVE to execute the SMP/E receive command for the z/XPF function.

7) In the install dataset run member APPLY to execute the SMP/E apply command for the z/XPF function. The apply command populates the target libraries with the z/XPF load modules, panels, messages, clist, proc, etc.

8) This step is optional. In the install dataset run member ACCEPT to execute the SMP/E accept command for the z/XPF function.

After installation (SMP/E or not) you still have to:

Here is a summary of the next four steps you'll need to complete in order to install z/XPF. They are covered in greater detail below.

1) Get the started task JCL into a procedure library.
2) Get the ZXPFCLST moved to a CLIST library.
3) Copy the ISPF panel, messages and table libraries.
4) Define z/XPF to your DB2 system (see the topic "z/XPF and DB2" below).

That was the summary. Here are the details.

1) Get the started task JCL into a procedure library.

Within &HLQ.INSTALL.JCL is the skeleton JCL member needed to execute the z/XPF started task, ZXPFCPTR. Below is a copy of the JCL contained within this member.

```
//*
//ZXPFCPTR EXEC PGM=APBEGN,REGION=0M
//*
//STEPLIB  DD  DSN=&HLQ.APFLOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//INPUT    DD  DSN=&HLQ.CNTL(INPUTDS),DISP=SHR
//ZXPFLOG  DD  SYSOUT=*
//SYSABEND DD  SYSOUT=A
```

Copy the member to a proclib dataset as appropriate to your environment. Modify the dataset name on the STEPLIB DD statement to conform to the library you will be using to execute the z/XPF started task. Modify the dataset name on the INPUT DD statement to conform to the library you will use for the input control statement(s) the started task uses during initialization. The STEPLIB DD library, &HLQ.APFLOAD, *must be an authorized library.*

2) Get the ZXPFCLST clist moved to a clist library.

The 2nd member of &HLQ.INSTALL.JCL you will need to move is ZXPFCLST. This is the clist used to allocate the z/XPF libraries, and invoke the ISPF interface. Below is a copy of the first few lines of code within the clist. You must modify these lines within the clist to specify the correct libraries. Probably a good idea to shorten the name of the clist at this time from ZXPFCLST to ZXPF.

```
PROC 0 DEBUG LIST PRM('ZXPF') +
     LLIB(''&HLQ.APFLOAD'' +
     ''&HLQ.APFLOADZ'') +
     MLIB(''&HLQ.ISPMLIB'') +
     PLIB(''&HLQ.ISPPLIB'') +
     TLIB("&HLQ.ISPTLIB")
```

&HLQ.APFLOAD should be set to the load library that contains modules APISPF, APDSNMDS, and APD2STMT. Change the "&HLQ" value to the value you chose for this install of z/XPF.

&HLQ.APFLOADZ should be set to the load library that contains module COMPRES2. Change the "&HLQ" value to the value you chose for this install of z/XPF.

&HLQ.ISPMLIB, &HLQ.ISPPLIB and &HLQ.ISPTLIB should be changed to the appropriate library names.

Note that if the system name is changed from the default value of ZXPF, the parm passed to the clist to at startup will also need to be changed.

3) Copy the ISPF panel, messages and table libraries.

At this point the ISPF panels needed are in dataset &HLQ.ISPPLIB and the messages are in &HLQ.ISPMLIB. You can either execute the ISPF clist using these libraries, or copy the members to an appropriate library for your environment.

z/XPF and DB2

If z/XPF is to be used to profile DB2 applications, then z/XPF's DBRM must be bound to the target DB2 systems. This DBRM will give z/XPF the ability to gather package and plan

information for SQL.

z/XPF's DBRM accesses the catalog, and augments SQL performance data with package bind and dependency statistics that enhance z/XPF's profile reporting.

To bind z/XPF's DBRM, modify and run the BINDJOB member of the install library on each DB2 system where z/XPF will be used. The user executing this job must have BIND ADD authority within the DB2 system. SYSADM works as well. This step must be done for each DB2 system that z/XPF will be deployed to profile.

Finally, the userid that the z/XPF server address space uses will have to be given authority to execute the plan. The easiest way to accomplish this is through SPUFI. Assuming the server task name is ZXPFCPTR and it defaults to a user id of ZXPFCPTR, the following grant sql statement will give z/XPF the ability to execute the plan that was bound above.

GRANT EXECUTE ON PLAN APGETSQL TO ZXPFCPTR

This GRANT SQL statement will need to be successfully run on the same DB2 systems where you bound the plan. The userid that binds the plan will have ownership of the plan and should be able to grant execute authority. If not, DB2 SYSADM authority will be needed.

z/XPF and Installation Security

During a data capture, z/XPF allocates and writes to a VSAM ESDS dataset. The format of the dataset name is "&HLQ.ADDRESSSPACE-NAME.DMMDDYY.THHMMSS.PROFL", where:

&HLQ can be one of the following three values:

1) z/XPF's subsystem name (defaults to "ZXPF"), or
2) A specific qualifier chosen by the user, and specified in z/XPF's start-up control statements, or
3) The TSO userid of the individual who defined the data capture request.

Note: Depending upon the choice of a high-level qualifier, the installation may need to define an alias pointer in the Master Catalog for this high-level qualifier. Failure to do this *could* result in capture datasets being catalogued in the system's Master Catalog.

ADDRESSSPACE-NAME is the name of the batch job, Started Task or TSO session *that is the target* of the data capture request.

DMMDDYY is the current date:

"D" - a constant
"MM" - the month

"DD" - the date of the month
"YY" - the year

THHMMSS is the time:

"T" - a constant
"HH" - the hour
"MM" - the minute
"SS" - the second

PROFL - a constant

Whether you intend to have a specific high-level qualifier for all data capture datasets, or you intend to use the requestor's TSO userid as a high-level qualifier, the z/XPF server task needs ALLOCATE authority to create these datasets.

[The first time a dataset is opened for report generation, z/XPF's mapping process will create csect maps for all Private Area load modules. These maps are appended to the data capture dataset when the user frees the dataset. Therefore, all TSO users that wish to process data capture datasets will need UPDATE authority to them.]

The z/XPF user has the option of utilizing the installation's security product to control two functions:

- The user's ability to schedule a data capture session for a target address space, and
- The user's ability to use GTF SLIP processing to add PER event data to the capture dataset.

NOTE: z/XPF's ability to use SLIP processing is, by default, off. To ensure that z/XPF users can add PER data to their data capture dataset, z/XPF must be initialized with slip commands turned on. You can do that by specifying the following statement in the startup deck:

SLIP_COMMANDS=YES

z/XPF's ability to add PER event data to the capture dataset allows the user to collect Branch TRACE and INSTRUCTION FETCH information. The result is very useful, extremely detailed, and yields what we call "High Definition Profiling". However, using the PER feature results in significant CPU consumption, which the installation can control through the application of security rules.

z/XPF can use generalized resource rules to control these two functions. During initialization a RACROUTE request is executed to list rules for "&HLQ.**". The "&HLQ" value can be a default of "ZXPF", or specified in the input control statement "RACF_PROFILE=" (as detailed below). The RACROUTE request builds a list of profiles in the z/XPF address space that begin with the "&HLQ" value, if any exist.

When a user adds a data capture request to one of z/XPF's queues, a RACROUTE REQUEST=FASTAUTH is executed for rule "&HLQ.ADDRSPAC.&NAME", where "&HLQ" is either the default value of "ZXPF" or as specified in the "RACF_PROFILE=" control statement. "&NAME" is the name of the address space that is the target for the data capture request.

If the return code from the RACROUTE REQUEST authorization check is 0 or 4, the request is added to the queue. Anything greater than 4 causes the request to be denied. A return code of 0 indicates that the user has READ authority to the rule. A return code of 4 indicates that the requested rule is not defined.

After passing the above check, if the data capture request specifies that GTF is to be used to add PER event data, then another RACROUTE is built and executed. The entity checked on this request is &HLQ.OPERCMDS.&NAME.

If a data capture request is denied after updating the correct rules, then z/XPF's copy of the security rules must be refreshed. The user has two options: Either re-cycle the z/XPF address space, or enter the operator modify command 'F &ZXPF,REFRESH PROFILES' where &ZXPF is the name of the z/XPF server address space.

z/XPF's use of SMF Exits IEFUSI and IEFACTRT

z/XPF uses the IEFUSI exit to notify the z/XPF server address space that a jobstep has started and uses IEFACTRT exit to notify the z/XPF server address space that a jobstep has terminated. Therefore, these two SMF exits should be defined to the system in the SMFPRMxx member. If these exits are present, then z/XPF will dynamically install exit code into the list of modules called at these exit points.

z/XPF checks for these exits at both the SYS and SUBSYS level. SYS-level exits take precedence for the entire system UNLESS SUBSYS-level exits are present, in which case the SUBSYS exit will take precedence for that subsystem (only).

An example of the SYS-level exit would appear thusly:

SYS(EXITS(IEFUSI,IEFACTRT))

An example of the SUBSYS-level exit for JES2 would appear thusly:

SUBSYS(JES2,EXITS(IEFUSI,IEFACTRT))

So, if a SUBSYS exit exists for JES2/JES3 subsystems, BUT no SYS-level exit is present, then z/XPF will be able to measure start-of-job/end-of-job for JES2/JES3-submitted jobs only, and will not be able to measure Started Tasks, or TSO sessions. Thus, it is highly desirable to define these exits at the SYS level AND at the SUBSYS level for all desired sub-systems so that z/XPF can use these exits.

To verify the presences of these exits, the user may execute the Operator command: "D

SMF,O". This command will display the status of SMF on the local installation. These two statements must be present:

SYS(EXITS(IEFUSI)) -- PARMLIB
SYS(EXITS(IEFACTRT)) -- PARMLIB

If either of these two statements are not present in the display, z/XPF will initialize, but will generate messages upon each startup to remind the user that these exits are not available to z/XPF. In this case, z/XPF will not be able to measure start-of-job/end-of-job events.

Short running jobs cannot be profiled with z/XPF

z/XPF should not be used to profile jobs that run in only a matter of seconds, as data capture initialization may not complete before the job completes.
Here is an explanation:

z/XPF needs to scan the target application control blocks to identify JOBLIB and/or STEPLIB dataset information. The datasets concatenated to these DD statements are used to map load modules identified during data capture to allow reporting at the csect level.

The IEFUSI SMF exit installed during server initialization is used to communicate to the z/XPF server address space that a job on the start-by-jobname queue is starting. It is possible for the z/XPF server be in the midst of initialization while the target application has already ceased execution.

To allow for the possibility that the target application has not been given control yet (perhaps because z/OS still needs to finish initialization before it can run), z/XPF doesn't even look for the JOBLIB/STEPLIB information until after the 200th interval. If z/XPF's default interval rate is taken by the customer (via the control statement, "NBR_COPYCYCLES_PER_SECOND=nnn), this means the server won't even look for that information until after a minimum of 4 seconds.

zXPF and prior version compatibility

Over time, z/XPF's internal structures have evolved considerably, and will continue to do so. Therefore it is possible that new versions of the product will no longer process data captured under older versions of z/XPF.

It is unlikely that customers will ever "archive" data capture datasets except in certain circumstances, so we feel the exposure is minimal. However, over the long term logic will be added so that z/XPF can react and adjust in its processing of data captured under previous versions of the product.

Of course, it is possible to retain older versions of the z/XPF product in "inventory", and these

older versions could be used to work on older data capture datasets. For the present, this is an acceptable work-around for the problem.

z/XPF and virtual storage

z/XPF pre-allocates virtual storage at initialization time in order to greatly increase its efficiency during execution. In environments where virtual storage is constrained, z/XPF may fail to initialize, or experience problems during operation. Be advised that systems personnel may need to relax virtual storage limitations in order to receive the full benefit of z/XPF's abilities.

Note: For z/XPF Release V1R1 and above, z/XPF utilizes 64-bit storage (above-the-bar) for data collection and processing. Therefore it is recommended that the installing Systems Programmer set the MEMLIMIT value in the SMFPRMxx member to NOLIMIT.

Setting the default service class designation for z/XPF

z/XPF needs a certain level of execution "priority" in order to gather data efficiently. Therefore, it's a very good idea to set the proper service class designation so that Workload Manager assigns enough execution time to z/XPF. Otherwise data capture datasets will be incomplete, and the resulting reports will not be accurate or may even fail.

The default z/OS service classes for started tasks have these definitions: "SYSTEM", "SYSSTC", and "STCLOM". "SYSTEM" gets the most service, "SYSSTC" gets less service than "SYSTEM" and "STCLOM" gets the least service.

z/XPF's default service class is SYSSTC (provided that the installation does not prevent program-generated Operator commands). If necessary, an Operator command can be issued to reset the service class for the z/XPF server task after z/XPF initializes. This command is:

E ZXPFCPTR,SRVCLASS=XXXXXX

This Operator command will put z/XPF into the service class defined as "XXXXXX", whatever that value is for the local installation.

z/XPF's service class can also be influenced by an input control statement in z/XPF's startup "deck". That statement is:

RESET_SRVCLASS=XXXXXX

If this statement is present, then z/XPF's initialization logic will execute the RESET command to set the server task to the service class "XXXXXX".

Again, *z/XPF's reports are only as good as the data it can capture*. Setting too low a service class will result in z/XPF not being dispatched often enough to gather the data users need in order to use the product effectively.

z/XPF's Started Task Name

The name of the member in the procedure library may be changed by the user. The default is ZXPFCPTR.

z/XPF Server logs its actions in ZXPFLOG:

In operation, z/XPF keeps track of its actions in the ZXPFLOG dataset. The dataset is allocated as a JES SYSOUT dataset. The DD statement used is ZXPFLOG. You can use this log to verify what's going on. If you experience unusual results or problems, then please retain the ZXPFLOG dataset for transmission to us for problem determination. You can find it in the started task JCL for z/XPF.

z/XPF and LPA mapping functions

During z/XPF's initialization process it creates Binder maps for modules that are located in the Link Pack Area (the "LPA"). This process can take several minutes and can consume a large amount of CPU. Most z/XPF customers want z/XPF to be able to reference modules in the LPA, but if your installation does not want to allow this as a default action for z/XPF, then please include a MAPLPA=NO control card in your z/XPF startup deck.

z/XPF and Operator commands

Installing systems programmers may need to interact with system security personnel in order to secure the authorization for z/XPF to issue three Operator commands that it needs in order to do its job.

In order to set the correct service class for z/XPF, it issues:

RESET <server-name>, SRVCLASS=SYSSTC

In order to minimize "wrapping" of the system trace tables, z/XPF issues:

TRACE ST,nM

"nnnK" refers to the size of a trace table in Kilobytes. "nM" refers to the size of a trace table in megaybtes.

z/XPF can also be configured to issue SLIP SET and SLIP DELETE Operator commands at the user's request. When SLIP is set by z/XPF, Instruction Fetch or Branch Trace information is recorded in the capture dataset. This is a very good way for z/XPF users to "drill down" to the lowest level in their analyses, and we recommend that this ability be granted to z/XPF's users.

If the installation decides to grant SLIP usage to z/XPF users, then security policies may need to be reviewed and/or altered so that z/XPF users may issue the appropriate Operator commands detailed below.

When the SLIP command is processed on z/XPF's behalf, the command is "fire and forget" from z/XPF's point of view, and no check is made to see if the command was executed successfully. If a z/XPF user does not obtain the results he or she expected when the attempt to use SLIP was made, the installation's security policy may be the "culprit".

z/XPF also has two optional control card statements that affect its SLIP processing:

SLIP_COMMANDS=YES/NO (If the card isn't present, the default is "no")
and
SLIP_ID=xxxx (where "xxxx" is the subsystem name. The default value is "ZXPF")

For complete information on these control statements, please read the section entitled, "z/XPF's Control Statement Explained" below.

Format of the SLIP commands that z/XPF may issue

To turn PER on for instruction fetch:

SLIP SET,I,RANGE=(low-addr,high-addr),ACTION=STRACE,JOBNAME=jobname,ASID =yyyy,PRCNTLIM=99,ID=xxxx,END

To turn on PER for successful branch trace:

SLIP SET,SBT,RANGE=(low-addr,high-addr),ACTION=STRACE,JOBNAME=jobname, ASID=yyyy,PRCNTLIM=99,ID=xxxx,END

Explanation of the variables used in the SLIP commands above:

low-addr and high-addr are virtual storage values that z/XPF "plugs in" automatically. These values are computed based upon the CDE information for the loaded module, and the information specified by the user in z/XPF's ISPF panels about which csect and offset within the desired load module the user wishes to monitor.

=jobname is the name of the targeted address space.

yyyy is the ASID value of the target address space.

xxxx is the SLIP ID. If the SLIP_ID control card is not invoked for this instance of z/XPF, then the default is "ZXPF". Otherwise the value in the SLIP_ID control card is used.

A third, and final SLIP command is issued by z/XPF to turn off PER. It is:

SLIP DEL,ID=xxxx

In this command "**xxxx**" will match the "ID=" field on the SLIP SET command.

Caution: z/XPF's SLIP processing can be costly in CPU cycles, and so the decision to use SLIP processing under z/XPF should be considered wisely. However, this facility can be VERY useful in certain situations where "high-definition profiling" is desired. There is no better way to drill down to such a granular level of detail.

Again, we recommend that z/XPF's users be granted the authority to use z/XPF's SLIP Facility. We also recommend that this feature be used with caution and due understanding of the costs of such CPU-intensive activity.

Finishing the z/XPF installation

1. The z/XPF load library used for the started task must be authorized. Either add the target load library to the authorized list, or copy these modules to an authorized library:

   - APGETSQL
   - APLOGT
   - APMAPP
   - APMERGE
   - APMRGECT
   - APMVTTE
   - APNTVCT
   - APPBUFCP
   - APPBUFCT
   - APRESMGR
   - APTBUFCP
   - APTBUFCT
   - APVSAM
   - MERGCHNS
   - PBUFCOPY
   - TBFZOS10

2. Copy member ZXPFCPTR from the &HLQ.INSTALL.JCL library to a PROCLIB on your z/OS system.
3. Modify the STEPLIB DD to point to the load library where the z/XPF modules are located.

   NOTE: z/XPF must run as a started task, not as a batch job

4 Create the file that will hold z/XPF's control statements.

   a. Create a sequential, fixed-block 80-byte file or a member of a PDS.
   b. Syntax rules: Control statements must begin in column one, with each different statement type on a separate line.
   c. While you may choose to include any of the control statements included below (in “Input Statements for z/XPF’s Started Task”), you MUST include the License Code statement. It must NOT be omitted.
   d. To input the License Code Statement, enter the characters “LC=”, then paste the activation code that you got from Duke Software.
   The string (beginning with “LC=”) MUST appear in column one of the line.
5. Close the file.
6. Copy member ZXPFCLIST from the &HLQ.INSTALL.JCL library to a valid CLIST library.
7. Modify the value in the PRM variable to the sub-system name intended for use by this version of z/XPF.
8. Modify the LLIB, PLIB, and MLIB statements to point to the target libraries.

Input statements for z/XPF’s started task

When z/XPF’s server address space is started, z/XPF looks for the INPUT DD statement. The DD statement points to a file that contains control statements for z/XPF’s current instance.

Note that it is possible to run multiple instances of z/XPF. In the unlikely event that this becomes desirable, the SSNAME statement must be used, and each SSNAME statement’s contents must be uniquely named and different from other instances of z/XPF.

z/XPF control statements explained

LC=xxxx-xxxx-xxxx-xxxx

Where the parameter “xxxx-xxxx-xxxx-xxxx” is a 20-character activation code supplied by Duke Software. **This control statement is mandatory. z/XPF will not initialize without it.**

When a new trial of z/XPF is requested, a start and end date is negotiated by management at the installing customer site. When z/XPF has been installed and if the current date is before the formal start date of the trial, z/XPF will run, but with a limitation on the number of events it will capture for any data capture job. This is so the installing Systems Programmer can verity z/XPF’s proper installation. When the formal start date arrives, then z/XPF will run without restriction.

That’s the most important control statement, so we put it first. The rest appear in alphabetic order.

ALOCVOL=xxxxxx

Use this control statement to place profile data capture datasets on a specific volume. With this statement set, the dynamic allocation routine constructs the parameter list requesting the allocation on this volume.

There is no default value. When specified, the parameter “xxxxxx” is any valid DASD VOLSER within the installation.

If this control statement is not present, then the storage request will be satisfied using whatever storage management rules are already in place.

ALOCUNIT=xxxxxxxx

If not present, ALOCUNIT defaults to the installation’s default unit type. If stated, this control statement will direct allocation of data capture datasets to a specific unit or generic unit type. ALOCUNIT accepts a 1 to 8-character string. No validity checking is done on this character string.

DATA_CAPTURE_DSN_HLQ=xxxxxxxx

“xxxxxxxx” specifies a one- to eight-character string to be used as the high level qualifier on profile data capture datasets allocated by the started task. Any characters that are valid for a dataset name may be used.

If this control statement is not present, then the high level qualifier defaults to the sub-system name. At present, z/XPF will accept a maximum of eight characters for this value. We will expand this limitation in future releases of z/XPF.

If the character string of “USERID” is entered as the parameter for this statement, then the started task will allocate the capture dataset using the TSO USERID of the individual that scheduled the profile capture request. If this choice is made, then the user must ensure that the z/XPF started task has the authority to allocate and open for output datasets with those userids.

To reduce administrative overhead, set this value, and then give individual users authority to the datasets created by their profile capture requests.

DATA_CAPTURE_DS_BUFFERS=nnn

This Control Statement establishes the number of buffers z/XPF uses during data capture. If this control statement is not present, the default value is “15”. The higher the value, the

more virtual storage is used to hold the buffers, but the number of I/Os executed to write to this dataset is reduced. The use of this control statement is highly recommended.

If specified, the parameter “nnn” is a numeric value valid for the BUFND setting for an ACB. The specified value is used with the profile data capture dataset. The block-size for this dataset is 4K.

DB2=XXXX,SDSNLOAD=YYYYY.ZZZZZZZ

This control statement is highly desirable for measuring DB2-related programs. XXXX is a version identifier for a version of DB2 (The version identifier used to be a three character value. For DB2 Release 10 and above, it is a four-character value). YYYY.ZZZZZZ is the dataset name for that version’s SDSNLOAD Library. Below is an example of this control statement for DB2 Version 8.1:

DB2=810, SDSNLOAD=DSN810.SDSNLOAD

The presence or absence of this control statement indicates to z/XPF that DB2 catalog information for DBRMs, Packages, and Plans should be acquired and added to the data capture dataset. z/XPF will then use the Call Attach Facility to connect to the target DB2 system identified during data capture.

For each version of DB2 present on the target system one statement is needed. Multiple DB2 systems at the same version level can share the same SDSNLOAD dataset.

For DBRMs bound into packages, DB2 catalog tables SYSIBM.SYSPACKAGE and SYSIBM.SYSPACKDEP are queried for bind and dependency information, and SYSIBM.SYSPACKSTMT is queried for SQL text. In order for z/XPF to access these catalog tables, the DBRM shipped with z/XPF must be bound on the target DB2 system. Also, z/XPF must be allowed to access the DBRM via installation security systems.

When a query of SYSIBM.SYSPACKAGE returns a “not found” condition for a DBRM, catalog tables SYSIBM.SYSPLAN, SYSIBM.SYSPLANDEP, and SYSIBM.SYSSTMT are accessed.

FORCE_VENDOR_TABLE=YES

This control statement will store z/XPF’s common area data block in the vendor table anchor entry assigned by IBM for use by z/XPF. It is only to be used in circumstances wherein (for some reason) z/XPF’s vendor table anchor slot has become corrupted (meaning the slot contained non-zero values but also did not contain z/XPF’s data area).

INTERVAL_TOLERANCE_PERCENTAGE=nn

The parameter “nn” is a percentage value. The default value is 10 percent.

In a heavily loaded environment it is important to verify that z/XPF is capturing ALL the data available, without lapses. The INTERVAL_TOLERANCE_PERCENTAGE is used to compute whether z/XPF is getting control often enough.

The “Interval Rate” is the number of times per second that z/XPF gets control of the processor to scan trace records. The Interval Tolerance Value of “nn” is used as a percentage to calculate whether the achieved interval rate is within range of a “desired” interval rate.

[*For example, on our development system, we may expect to get fifty “intervals” per second, and at the end of ten seconds, we’d expect to see a total of 500 intervals. However, we may not be able to achieve exactly that number of intervals, so a ten percent “tolerance” value would be used. If the Interval Rate fell below 450 in ten seconds (90% of the “desired” Interval Rate) z/XPF would begin to generate messages.*]

Put in other words, if the achieved Interval Rate is greater than the desired Interval Rate minus the tolerance value then all is well.

After the first ten seconds of execution, z/XPF compares the achieved Interval Rate with the actual Interval Rate and computes it against the Interval Tolerance percentage. If all is well, z/XPF checks again twenty seconds later. If all is still well, z/XPF checks again thirty seconds later. If an exception is seen, z/XPF will generate messages to the session log and the 10-20-30-second monitor cycle will begin again.

MAP_LMOD_DURING_CAPTURE=YES/NO

z/XPF normally performs mapping functions DURING data capture. This causes the z/XPF server address space to allocate and open datasets that reside in Joblib/Steplib and Linklist as input to the Binder. This may cause a security problem for the z/XPF Server Address space. If so, you can turn off this behavior by adding the above Control Statement with a parameter of “NO” to turn mapping off during data capture. Mapping can then be done later, during the report generation phase.

If this control statement is not present, then the default value of the parameter is “YES”.

MAPLPA=NO

By default, z/XPF will attempt to map modules in the Link Pack Area (LPA) in order to create BInder Maps. Depending on the number of modules in the LPA, this process can take several minutes and consume a large amount of the CPU. This defaul action can be over-ridden by includeing the MAPLPA=NO control statement in z/XPF's startup deck.

MAP_LPAMOD=xxxxxxxx,DSN=yyyyy

Use this control statement to inform z/XPF of the location of LPA resident Load Modules that are not mapped during z/XPF's normal initialization. z/XPF will thereafter use this information when calling the Binder to create Csect maps for the named Load Module.

In this control statement, xxxxxxxx should contain the Load Module name, and yyyy the dataset name to be used for the Binder dialogue.

You may define no more than 100 MAP_LPAMOD statements to z/XPF.

z/XPF's initialization logic uses the LPAT table, mapped by Dsect IHALPAT as the source for the calls to the Binder. When the mapping logic has processed the last dataset in the LPAT, and there are LPA resident modules not yet mapped, message XPF000E-03 is written to the ZXPFLOG for each Load Module not yet mapped. These modules could be "candidates" for this special mapping function. If you wish to have z/XPF map these modules, add a MAP_ LPAMOD control statement for each module.

MAX_MSG_DURING_CAPTURE=1000/NNNNNN

This parameter sets an upper limit on the number of messages generated by z/XPF's server address space. It prevents z/XPF's data capture from writing redundant messages to the z/XPF log. If this number is exceeded, then z/XPF may be in a loop, and all active data capture sessions are stopped.

If this control statement is not present, the default value of the parameter is "1000". This can be set to any value desired up to 999999. It may be overridden by keying in all zeros, thusly: "000000" In that situation, there will be no limit on the number of messages logged.

NBR_COPYCYCLES_PER_SECOND=50/nnn

The parameter “nnn” is used to compute the target for the number of times per second z/XPF will scan for events. At the beginning of each interval, z/XPF notes the time the interval started. When all of the interval processing is completed, the start time for the next interval is computed, and the current time is subtracted from the next interval start time, to give the pause time.

If this control statement is not present, the defaults value is “50”. If specified, “nnn” is any value between 1 and 100.

If events occur that z/XPF doesn’t seem to capture, then it is possible that this parameter has been set to a value that is too low.

PR_BUFFERS=nnn

In order to minimize “wrapping” of system Trace Buffers (and subsequent data loss by z/XPF), z/XPF can adjust the number of processor Trace Buffers by executing the “TRACE ST” Operator command during z/XPF’s initialization. The larger the Trace Buffers, the greater the chance that z/XPF will be able to keep up with execution in very fast-throughput environments.

z/XPF will enter the commands thusly: “TRACE ST,nM”

For z/OS V1R10 and above, the system default is 256 4K buffers for a trace table of 1 megabyte. In these systems, z/XPF will set the processor trace table to 512 buffers per processor, for a trace table of 2 megabytes. The maximum value z/XPF will accept on this statement is 1280, or 5 megabytes per processor. Any value greater than that will be set to 1280. Any value less than 256 will be set to 256. Any value entered in the control statement that is less than the currently set amount will be ignored.

PR_BUFFERS=ASIS

If this statement is present, z/XPF will not adjust the trace table size. Be advised that this may prevent z/XPF from capturing all of the trace records it needs in order to do its job.

RACF_PROFILE=”hlq”

If specified, the parameter “hlq” is the High Level Qualifier for a set of RACF security profiles. If unspecified, the value is a default of “ZXPF”.

Later, during initialization, a RACROUTE is executed to create a list inside the z/XPF address space of the profiles that are relevant to z/XPF.

When a request is to be added to one of the queues, a RACROUTE authorization check is

made. The entity used for the check will use the parameter given in the RACF_PROFILE statement.

Potential issue: As long as a user does not specify a RACF_PROFILE= statement in any instance of z/XPF, or specifies the same value in all instances, all is well. HOWEVER, if one instance of z/XPF specifies a RACF_PROFILE statement, and then the installation starts another instance of z/XPF, but DOES NOT specify the same high level qualifier for that instance, *then that instance of z/XPF will run un-protected.*

The easiest course of action is to take the default, or specify the same value in the RACF_PROFILE= statement on all instances of z/XPF.

RESET_SRVCLASS=XXXXXX

If this statement is used, it will cause z/XPF to issue an Operator RESET command to set z/XPF's service class to the named service class. The default value for z/XPF is "SYSSTC".

RESTARTDSN=dsname

If specified, the parameter "dsname" is the name of a dataset. This dataset is used to hold profile requests in the z/XPF started task queues when the z/XPF server terminates. During start-up it is allocated and read. All requests in the start-by-jobname queue are restored. Any request in the start-by-time queue that has not expired is restored.

If the RESTARTDSN statement is used, then the parameter "DSNAME" dynamically allocates a dataset of that name (if it is not already present when the task initiates) as a physical sequential, fixed block file, with a block-size of 9600. It is a one-track dataset, with a one-track secondary allocation.

SLIP_COMMANDS=YES/NO

If this control statement is not present, the default value of the parameter is "NO".

Specify "YES" to allow profile data capture sessions to include SLIP PER interrupts. z/OS allows only one SLIP of this type to be active at a time.

When an active profile capture session contains a request for SLIP records, z/XPF compares the identified load modules in the target profile address space to the load module name in the user's request. When a match occurs, a SLIP command is created and sent to z/OS via MGCRE (SVC 34) if z/XPF does not already have a SLIP active at that time for another profile session.

When the profile data capture session terminates, z/XPF checks to see if a SLIP was issued. If it was, it constructs another command to terminate SLIP processing, and submits that to z/OS again using MGCRE.

SLIP_ID=xxxx

If this control statement is not present, z/XPF will default to using the sub-system name given in the SSNAME statement. Specify any set of one- to four-characters that are valid for the ID used in a SLIP command. Note: this ID should be unique among SLIP IDs used in the installation. That is, pick a SLIP ID that will ONLY be used by z/XPF.

SSNAME=ZXPF/xxxx

The value specified here is used as the z/XPF sub-system name. If this control statement is not present, the default value is “ZXPF”.

Many instances of z/XPF may run concurrently, but each must be uniquely named. If specified, the parameter “xxxx” can be any one- to four-character name.

SSCLEAR=YES/NO

If this control statement is not present, the default value for the parameter is “NO”.

Use this statement with caution. If stated, this statement will cause z/XPF to clear the Subsystem Control Table for a previous instance of z/XPF with the same name as given in the SSNAME parameter. This is useful in situations where a previous instance of z/XPF has terminated abnormally.

If there is NOT another z/XPF server address space active using the name specified by the SSNAME control statement, then setting this to YES cannot do any harm. HOWEVER, when another z/XPF server address space is active and is using the same SSNAME parameter, using SSCLEAR=YES will cause errors within the other active server address space.

USER_TRACE_NBR=0-F

If this control statement is not present, the default value of the parameter is “F”.

This control statement is used in conjunction with the ZXPTRAC “User Trace” feature. ZXPFTRAC allows you to create your own “Trace Records” that are written to system

Trace Buffers and later reported on by z/XPF. The feature allows you to signal events in complex code flows,

For example, you could execute a ZXPFTRAC with TYPE=BEGIN into your code prior to scheduling it to run on a zIIP processor, and then execute a ZXPFTRAC with a TYPE=END once the code is running on the zIIP processor. In this way, you can measure the amount of system overhead it takes to get your code ported over to the zIIP processor.

WRITE_TO_LOGREC=YES

This control statement is a diagnostic tool for special circumstances. It is used to force the writing of log records when trouble-shooting z/XPF's SRB-based data capture logic. Specify this control statement only at the direction of z/XPF's Technical Support personnel.

Installation verification/trouble-shooting

Duke Software does not supply any installation verification mechanism. Too many variables can affect the data capture function for us to be able to supply a canned verification process. If you have taken care of whatever security concerns/issues there may be in your environment, and you are confident WLM will treat the z/XPF address space correctly, try starting the address space. A reminder: z/XPF must execute as a started task, not as a batch job.

If z/XPF doesn't come up, check the ZXPFLOG SYSOUT dataset for the address space to see what the error messages indicate.

Once the started task initializes, execute the CLIST you copied to the CLIST library.

Select Option 1 from the z/XPF Primary Option Panel to create a profile data capture request. If you have a short batch job you can run, enter a request to start a capture session at the same time as when the batch job becomes active. This is a request that will be added to the start-by-jobname queue. Try to use a job that you know will execute for at least 15 - 30 seconds, if not longer. Alternatively, you may start a data capture session for an active address space.

When the capture session terminates (either because the batch job finished, or because you used the ISPF interface to stop it), check the ZXPFLOG SYSOUT dataset for messages related to your capture session.

In the ZXPFLOG SYSOUT dataset you should see messages at the start of the capture session indicating the dataset was allocated, and another message at the end indicating it was de-allocated. If you encounter an error message instead, contact Duke Software Technical Support team for help.

Here's a handy usage note:

z/XPF creates its profile from the events that are occurring on the z/OS image while the target profile address space is active. On a very lightly loaded machine, there may not be enough activity to generate enough events to get an accurate profile, and this situation is possible on a development system.

To get around this, you can specify that GTF SLIP PER events be created. You can do this while setting up the data capture request and z/XPF will take care of it, or you can do it manually by entering the SLIP command from a console.

Thanks again for installing z/XPF. We hope it serves you well. If we can be of any further assistance to you, please don't hesitate to contact us.

Once again, you can reach us here:

Dave Day : (281) 395-5570 or David.Day@duke-software.com