Application and Program Performance Analysis Using PEX Statistics on IBM i5/OS

- Find where to invest to improve job performance
- Find application "hot spots"
- Analyze job program flow

Rick Turner
Deborah Hatt
Jim Cook

ibm.com/redbooks
Application and Program Performance Analysis Using PEX Statistics on IBM i5/OS

July 2007
First Edition (July 2007)

This edition applies to Version 5, Release 3, Modification 0 of i5/OS (OS/400), 5722-SS1. It also applies to the IBM iDoctor for iSeries V5R3, PEX Analyzer Service Tool Build Level S00134-C00520

© Copyright International Business Machines Corporation 2007. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

Notices ....................................................................................................................... ix
Trademarks ................................................................................................................ x

Preface ....................................................................................................................... xi
What PEX Statistics can do for you ......................................................................... xi
Who can benefit from using PEX Statistics .............................................................. xi
The team that wrote this book ..................................................................................... xiii
Become a published author ......................................................................................... xiv
Comments welcome .................................................................................................... xiv

Chapter 1. Introduction to PEX Statistics ................................................................. 1
1.1 Using this book .................................................................................................... 3
1.2 What is in this book ............................................................................................. 3
1.3 What data does PEX Stats provide ..................................................................... 4
1.4 The parts: iDoctor, PEX Analyzer, and PEX Statistics ....................................... 5
1.5 Using PEX Statistics .......................................................................................... 5
1.6 What can I do with PEX Statistics ...................................................................... 6
1.7 Collection options overview ............................................................................... 7
  1.7.1 What are the PEX Stats collection modes ..................................................... 8
  1.7.2 When is the data collected ........................................................................... 9
  1.7.3 What’s collected for each event ................................................................... 10
  1.7.4 What are inline data and cumulative data .................................................... 10
  1.7.5 Why is the Collection Event Count so high ................................................ 11
1.8 PEX Stats Quick Start guide ............................................................................... 11
1.9 What tools should I use ..................................................................................... 12
  1.9.1 When to use PEX Stats ............................................................................. 13
  1.9.2 When to use other tools ............................................................................ 14
  1.9.3 Infrastructure comparison with Job Watcher and PEX Trace ...................... 15
1.10 Stats Data Analysis Examples list ..................................................................... 15

Chapter 2. Getting started: using Stats Flat ............................................................ 17
2.1 Obtaining PEX Analyzer .................................................................................... 18
2.2 Authority requirements ...................................................................................... 21
2.3 Installing PEX Analyzer .................................................................................... 21
2.4 Starting iDoctor .................................................................................................. 21
  2.4.1 Connecting to a system ............................................................................. 22
  2.4.2 Signing on to a system ............................................................................. 23
2.5 Starting PEX Analyzer ...................................................................................... 23
  2.5.1 iDoctor components Access codes ............................................................ 24
2.6 PEX Analyzer’s initial window .......................................................................... 25
  2.6.1 PEX Analyzer initial menu list of libraries ................................................ 25
  2.6.2 PEX data collections in a library ............................................................... 26
  2.6.3 Viewing the PEX Analyzer reports ............................................................ 26
2.7 Collecting PEX Stats data ................................................................................ 28
  2.7.1 What is in a PEX Definition .................................................................... 29
  2.7.2 Library selection ...................................................................................... 30
  2.7.3 Basic or Advanced path selection ............................................................... 31
  2.7.4 Select FLAT or HIERARCHICAL collection type ...................................... 32
  2.7.5 PEX Stats Collection Options specification: Basic Path ......................... 34
Chapter 3. Analyzing PEX Stats Flat data

3.1 PEX Stats Flat reports ........................................ 58
3.2 PEX Stats Flat Program default report view ............... 58
3.3 PEX Stats Flat view by Pgm/MI Instr: analysis .......... 62
   3.3.1 Inline CPU use: the default Stats Flat analysis data view .......................................................... 63
   3.3.2 Times called .................................................. 64
   3.3.3 Inline Elapsed Time ......................................... 69
   3.3.4 Cumulative Elapsed Time or CPU usec ................. 70
   3.3.5 Markers: program name analysis ......................... 70
   3.3.6 Markers: MI Complex Instructions analysis .......... 72
   3.3.7 MI Create xxx Instruction Analysis sample view .... 72
   3.3.8 Considerations for Selection, Start/End timing, and service programs ..................................... 73
3.4 PEX Stats Flat Job level default report view ............. 74
   3.4.1 DB file Full Open analysis and iDoctor view modification ................................................................. 75
   3.4.2 Program behavior analysis using Stats Flat Times Called values .................................................. 79
   3.4.3 Job resource usage summary view ....................... 82
   3.4.4 The ratio of Cumulative to Inline ....................... 83
3.5 Generating your own custom tables and graphs .......... 86
3.6 Summary of Stats Flat data analysis ........................ 86

Chapter 4. Collecting PEX Stats Hierarchical data

4.1 Preparing for collection ...................................... 90
4.2 Create a PEX collection ..................................... 90
   4.2.1 Basic path .................................................... 90
   4.2.2 Common menus ............................................. 92
4.3 Processing Stats Hierarchical data ........................ 96
   4.3.1 Coordinating test jobs and collection start/stop .... 96
4.4 Running Stats Hierarchical data analysis ................ 96
   4.4.1 Multiple job analysis selection ......................... 100
   4.4.2 Multiple job analyses and member name ............ 101

Chapter 5. Analyzing PEX Stats Hierarchical data

5.1 Viewing Stats Hierarchical data analysis output ......... 105
5.2 The default initial view ...................................... 106
   5.2.1 Call Level, Reference Id, Partial Count Status fields ................................................................. 107
   5.2.2 Viewing the Hierarchical Default Analysis view .... 113
5.3 Where Used analysis example 1: Ensure Object .......... 114
5.4 Hierarchical analysis tips ................................... 116
   5.4.1 Finding hot spots .......................................... 116
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CUA®  IBM®  System i™
Domino®  iSeries®  System/36™
DB2®  Language Environment®  WebSphere®
i5/OS®  Redbooks®
Integrated Language Environment®  Redbooks (logo)®

The following terms are trademarks of other companies:

Java, JDBC, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Expression, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.
Preface

What PEX Statistics can do for you

Would you like to obtain insight and information very quickly about what jobs and programs are running on your IBM® System i™ configuration and what their resource usage behavior is? PEX Statistics is one of the best tools available that can provide that information for you in a timely manner.

After an initial analysis, would you like to quickly find more detailed information about which programs are spending the most time and resources on your system? Once you have gone through the basic PEX Statistics information, you can, if necessary, expand your studies by collecting and analyzing additional in-depth job and program level information.

Use this information to understand more about the application’s processing behavior and where possible changes can be made. PEX Statistics analysis can lead you to meaningful results with a minimal investment of your time. The intent of this IBM Redbooks® publication is to provide guidance as to where and how to proceed.

The level of detail available through PEX Statistics includes user, job, thread, program, module, or procedure.

Note: Analysis of PEX Statistics data can help guide those who are responsible for analyzing and improving application job and program level performance. The audience for this book includes technical operations managers, programmers, analysts, designers, operations programmers, senior system operators, and other qualified performance consultants.

Who can benefit from using PEX Statistics

This IBM Redbooks publication is intended for use by those generally familiar with most of the iSeries® IBM-provided performance tools available through the i5/OS® operating system's commands and the additional cost Performance Tools for iSeries, 5722-PT1, licensed program.

i5/OS comes with a detailed program level performance data collection capability called the Performance Explorer (PEX). i5/OS commands supporting the collection include Add PEX Definition, Start Performance Explorer, and End Performance Explorer. One of the Performance Explorer (PEX) collection options is called Statistics (*STATS), which collects the program level performance statistics, including CPU usage, disk I/O activity, and the occurrence of certain i5/OS and System i microcode level events. The Print PEX Report function of 5722-PT1 provides a basic view of this *STATS data.

PEX Statistics provides a richer interface for collection and analysis of the *STATS performance data than is available through the i5/OS PEX command and the Print PEX Report output.
PEX Statistics is part of the iDoctor for iSeries set of software performance analysis tools and associated services that go beyond the capabilities of generally available performance related tools in evaluating the performance of your iSeries-based application jobs. PEX Statistics is used when the generally available performance tools are not able to clearly identify causes for performance concerns. PEX Statistics gathers detailed job level performance and program to program control flow information and provides analysis of this detailed data.

PEX Statistics is one of the iDoctor tools and is a key tool for analyzing detailed job program resource usage and control flow performance data. This book:

- Provides examples of using PEX Statistics operational modes.
- Provides PEX Statistics collected data file definitions and SQL query examples to analyze this data that provide information beyond that of the PEX Statistics built-in reports and drill-down information.

This book's objective is to make the performance analyst proficient in using PEX Statistics as a key tool in their performance analysis tool kit.

Note that the following i5/OS applications and middleware are beyond the scope of this book:

- Domino®
- Java™
- WebSphere® Application Server

Sources for performance analysis tools covering these applications include:

- Performance Management for IBM System i Web site
  
  
  Select the **Resource Library** link and view all the whitepaper and other resources available for performance under Domino, Java, and WebSphere

- Performance Trace Data Visualizer for System i (PTDV). This tool is based upon collecting Performance Explorer CPU profile (tprof) data and trace data. This data includes method entry and exit, Java, and heap events. For more information go to:
  

- The entire set of iDoctor for iSeries performance services and tools
  
  
  One of the Java specific tools included here is Heap Analysis Tools for Java.
The team that wrote this book

This book was produced by iSeries performance specialists working with the International Technical Support Organization, Rochester Center.

Jim Cook is a Senior Software Engineer at the ITSO, Rochester Center. He leads teams that produce iSeries Announcement presentation sets that are maintained on the iSeries technical support Web sites and presents at ITSO iSeries Forums internationally.

Rick Turner retired from IBM in 1993 and since then has worked as an independent consultant on iSeries performance. He has been involved with many aspects of iSeries and its earlier systems performance since 1980. He has taught performance classes, presented at COMMON, published a number of performance articles for iSeries NEWS and its predecessor publications, and has extensive performance analysis experience in the US and worldwide, especially in Europe and Southeast Asia. Prior to moving from the IBM GEM Region to the Rochester Development Laboratory in 1973 to work on S/38 development, he had programming assignments supporting IBM projects at USAF SAC, the NASA Gemini/Apollo project, US Army Logistics Management Systems project, the IBM System 3 CCP (Communications Control Program) product, State Farm Insurance home office, and the McDonnell Douglas Automation Center. He can be reached at rtaio1@attglobal.net.

Deborah Hatt has 20+ years experience with IBM in analyzing customer performance problems on iSeries-based systems. In order to sharpen her skills, she left IBM for a year to work for the CEO of a premier iSeries banking software house. One of her IBM mentors was Rick Turner, one of the authors of this book. She holds a first class honours degree in pure mathematics and theoretical statistics from the University of Westminster in London, England. She is proficient in RPG and SQL programming and SQL performance analysis. She spent five years working for the Rochester Support Center developing the iDoctor for iSeries advanced performance analysis tools and now works in the IBM Rochester Lab Services team. She can be reached at dhatt@us.ibm.com.

Our appreciation goes to the following people for their contributions to this project:

Ron McCargar, Larry Cravens, Sandy Chroomey, Dan Cruikshank, Ed Grazier, Brandon Rau, Tom Edgerton, Lloyd Perera, Dick Bains, Mike Cain, Darren Herrera, Pete Kinczel

A special thanks to Jay Kurtz and Jim Ranweiler whose foresight and technical abilities made this tool possible.
Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbooks publication dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Discover more about the residency program, browse the residency index, and apply online at: ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online Contact us review book form found at:
  ibm.com/redbooks
- Send your comments in an E-mail to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400
Introduction to PEX Statistics

The purpose of this book is to provide guidance to using PEX Statistics, a part of the IBM Service Tool “PEX Analyzer”.

**Note:** PEX is an abbreviation for Performance Explorer. PEX is a part of the i5/OS licensed program. It is a data collection tool that helps the user identify the causes of performance problems. PEX Analyzer is a fee-based tool that uses Performance Explorer's collection functions. PEX Statistics is a part of PEX Analyzer and is used to collect and analyze program run time data either system-wide or for specific jobs.

This chapter discusses the purpose and function of PEX Statistics and what it can help you do. It provides an overview of the type of information that PEX Statistics collects and summarizes the type of data views that can be generated from the collected data.

**Note:** Would you like to obtain insight and information very quickly about what jobs and programs are running on your System i and what their resource usage behavior is? PEX Statistics is one of the best tools available to determine that for you in a timely manner.

Would you like to find more detailed information about which programs are spending the most time and resources on your system? Once you have looked at a collection's basic PEX Statistics Analysis information, you can expand your search by additional analysis of more specific job and program level information and, if needed, by collecting additional data.

PEX Statistics information can be used to understand more about an applications’ processing behavior and provide guidance as to where possible changes can be made. With a minimal time investment, PEX Statistics help you achieve meaningful results. The intent of this book is to help guide you in how to proceed.
Analysis of PEX Statistics data can help guide those who are responsible for analyzing and improving the performance of application jobs and programs. The audience includes technical operations managers, programmers, analysts, designers, operations programmers, senior system operators, and other qualified performance consultants. Those who work with improving a system's run time environments know that in the general case the term *system performance* has different meanings at different times. In some situations, the meaning could be to improve the elapsed run time of one or more jobs. In another case, it could mean trying to reduce overall system resource utilization while maintaining or improving service levels.

**Note:** Terminology note: In this book, the shortened tool name “PEX Stats” is used throughout the book.

**What is new**
PEX Stats has been available since V3R6. Changes to PEX Stats analysis in V5R3 extended the level and amount of information available and made it much easier to find the information needed to research a performance situation. These changes made PEX Stats an even more capable and productive application program performance analysis tool.
1.1 Using this book

As you read and study this book, it is very useful to have access to an i5/OS system with iDoctor and PEX Analyzer installed on it. This enables you to do hands-on experimentation with data collection, analyzing and viewing the results, and modifying the view.

The PEX Statistics analysis functions provide comprehensive output, but they cannot anticipate your needs and provide insightful information in the proper order for every situation.

Understanding what you are looking at and determining what other data you want to see and knowing how to build your own customized view of the data are primary steps towards becoming proficient in analyzing PEX Statistics data.

One of this book's primary objectives is to give you a starting point to proceed from to find solutions to your system's performance challenges.

**Note:** Conventions used in this book include:

- Menu options and item selection: All described mouse actions are based upon use of a right handed mouse.
- SQL syntax usage: Unless otherwise specified, the SQL examples use the *SYS naming convention (that is, libraryname/filename) and no end-of-line character.

To use this document's library and file naming conventions syntax you can enter the SQL statement either:

- Using the Start SQL command (STRSQL) from an i5/OS command line with STRSQL NAMING(*SYS).
- Using the iSeries Navigator Database → Run SQL Scripts interface. If you are using this interface, you need to specify which i5/OS SQL syntax (*SYS = libraryname/filename or *SQL=libraryname.filename) is being used.

By default, iSeries Navigator uses the *SQL format. If you want to use the *SYS format (libraryname/filename), use the Run SQL Scripts window and select the Connections tab. Then select JDBC™ Setup → Format. Change the format to *SYS.

You also need to end each SQL statement with a semicolon (;).

1.2 What is in this book

There are two PEX Stats data collection options. One option collects what is called Stats **Flat** data; the other option collects Stats **Hierarchical** data. There is more information about these options in 1.7, “Collection options overview” on page 7. Flat data does not show program to program information; hierarchical data shows program to program call/return flow.

To summarize the contents of this document:

- Chapter 1, “Introduction to PEX Statistics” on page 1 presents an introduction to PEX, iDoctor, and PEX Stats capabilities.
- Chapter 2, “Getting started: using Stats Flat” on page 17 describes how to obtain, install, and start iDoctor, and how to set up and run a PEX Stats Flat collection.
- Chapter 3, “Analyzing PEX Stats Flat data” on page 57 discusses the analysis of PEX Stats Flat data.
1.3 What data does PEX Stats provide

PEX Stats collects data that represents a summary of the program and MI Complex instruction activity in selected jobs during a collection run.

See the “Glossary” on page 249 for a description of a MI Complex instruction and “i5/OS programs and MI Complex Instructions short descriptions” on page 236 for a list of more common i5/OS programs and MI Complex instructions.

This data can help you identify the high cost resource intensive programs and MI Instructions that ran and the jobs in which they ran.

Your system operators may be regularly holding or canceling a lot of these resource intensive application jobs.

It could be that users are not satisfied with transaction response time, the batch job turnaround time is too long, or important reports are often delayed.

You can use PEX Stats data to help you identify where the problems are and lead you to identify what changes are needed. These may significantly improve performance by making the job more efficient.
Once you have identified the jobs that have program with high resource use or suspected “bad habits”, use STATS HIER to see precisely what they are doing and how much resources are being consumed; the latter part may surprise you. You may find that there are some “hidden” high cost functions in them.

**Note:** In this book, unless otherwise specified, the term “program” refers to either a MI program or a MI Complex instruction. When referring to an ILE program, a program really refers to the program/module/procedure. When referring to an OPM program, there is only a program name; the module and procedure name are not applicable and are blank in the data.

### 1.4 The parts: iDoctor, PEX Analyzer, and PEX Statistics

This section provides a brief introduction about what iDoctor, PEX Analyzer, and PEX Stats are and for what they are used.

iDoctor is a PC client based program that provides the run time environment for different service tool plug-ins. It can be considered a bridge between the user and the server machine's PEX Statistics analysis tool's programs.

Two iDoctor related service tools are PEX Analyzer and Job Watcher. They provide the user with different types and multiple levels of performance collection and analysis capability, each of which is quite effective when studying run time environment behavior.

- PEX Analyzer's Statistics tool collects program activity, such as elapsed time, CPU time, and disk I/O counts, for each program within a job at each program call and return.
- Job Watcher collects job level run and wait activity related information about a time interval snapshot basis. For more information about Job Watcher, go to:
  

The Job Watcher and PEX Analyzer tools incorporate various analysis, graphics, and views that suit the needs of a wide range of technically qualified users.

Each tool has two parts: One is the “client side” and the other is the “server side”. Each tool’s “server side” function provides collection and analysis functions. Each tool’s “client side” is part of iDoctor, a Windows® based program that controls data collection and viewing to the server's collection, analysis, and presentation functions. iDoctor is included with both PEX Analyzer and Job Watcher as part of the deliverable package. Consider iDoctor as both tools' common management interface. PEX Analyzer is a separate performance service tool that contains a number of different functions, one of which is PEX Stats, which is what this book discusses. The primary function of PEX Stats is to collect, analyze, and present individual program performance information at the job level or higher.

### 1.5 Using PEX Statistics

Use PEX Stats to collect and analyze performance data that can help you identify where changes are needed in order to improve performance and help achieve acceptable levels of throughput and response time. The objective is to help meet the business' performance objectives. There are many ways to view and analyze PEX Stats data. iDoctor's GUI provides the ability to generate various table and graphical views from the data.
In some cases, other IBM performance oriented tools, such as Job Watcher or Collection Services, may have already been used to collect and, in some cases, analyze run time performance data. The findings may have identified certain jobs that are using a lot of CPU time, issuing a lot of disk I/O operations, encountering certain types of wait conditions, and, in general, are running longer than expected.

The next analysis step could be to use PEX Stats. By adding it to your data collection routine, you can extend the analysis (sometimes called “drilling down”) to identify which of the application programs or operating system programs are causing slowdowns. PEX Stats data analysis shows individual program elapsed run time and can help the analyst determine job run time. However, the data does not include anything that identifies specific types of waits. Analyze wait conditions using IBM iDoctor's Job Watcher or specific PEX Analyzer trace functions such as Disk I/O, Auxiliary Storage Management, or Task Switch.

1.6 What can I do with PEX Statistics

There are many applications that may occasionally experience increased use of system resources and corresponding increases in total job run time or transaction response time. If job level performance is periodically tracked and reported to people that can properly react to it in time, then it is much less likely to become a problem. However, in many shops, the job run time or response time tracking mechanism consists of listening for user complaints that might signal the possible recognition of creeping performance degradation. Often the result is that all of a sudden the applications are consistently running much longer, and it is reported to a committee that meets, looks over the available data (if there is any), and decides on the spot how to fix the problem.

Before using PEX Stats, the user should track performance using the system’s Collection Services to get the data and write their own set of reports to measure and compare throughput and resource consumption for key jobs. Following these tasks, the user should move on to PEX Stats, assuming you know what jobs to look at, or to Job Watcher to look for specific instances of job wait conditions and possible program involvement in them.

There are at least two ways to use PEX Stats to investigate these types of problems: proactive and reactive.

Proactively:
▶ Have someone who is aware of System i performance collection and analysis capabilities. They do not necessarily have to know everything there is to know about how to use the tool.
▶ Measure new applications before they go into production to determine base costs per transaction or other unit of work.
▶ Periodically measure critical application jobs and compare their job performance to baseline values.

Reactively:
▶ Wait until there is a problem (often this is when the users complain).
▶ Determine where the problem might be.
▶ Figure out what tool to use to validate the preceding point.
▶ Find someone that knows how to use it, both to collect the data and to analyze it.
▶ Determine what to do to apply possibly both short and long term performance improvement changes.
Which method to apply depends on the situation; sometimes they are effective and sometimes they are not.

Use PEX Stats data to determine:

- What percent of a job's run time is attributable to a specific piece of application software, middleware, or operating system modules?

- What programs are showing an increased amount of CPU usage per application unit of work?

  This requires information about “units of work” performed by the job(s) during the data collection. For example, a “unit of work” may be invoices, checks, line items, and so on, processed by a job. Defining how to do this is beyond the scope of this publication; however, it is a necessary part of managing a business process, part of which is tracking performance values over time.

- If your system is experiencing batch job time overruns or transaction slowdowns, it is possible that specific programs or other jobs are causing it.

  This does not imply that in all cases the programs are poorly designed or programmed. It could be that there are too many calls to a program or MI instruction and possibly they could be significantly reduced. In some cases, they might be eliminated (data base file Full Open/Close is a prime example and is discussed in this book). It is possible that some program is currently doing more disk I/O operations per unit of work than it used to. This is quite possible if the data base files continue to have new access paths built over them and there is a lot of file change activity that requires more index maintenance. This sometimes is caused by a job and program that is not part of some critical process but has a direct effect on how well those critical jobs run. One common situation in high volume data base processing is that a data base file has too many deleted records in it and the time to read its data continually increases until at some point it becomes a problem. There are other possibilities, some of which are discussed in this book.

Use PEX Stats data to get an overview of program and MI instruction performance. The view in Figure 3-1 on page 58 is a sample summary of program and MI instruction activity. This view and variations on it shows run time performance information that provides insight into the application program's overhead.

### 1.7 Collection options overview

There are two key PEX Stats data collection options. One of them tells the collection function what data collection mode to use, either “Flat” or “Hierarchical”. The other tells the collection functions which jobs to enable for data collection.

**Note:** PEX data can be collected with iDoctor, CL commands, or the System i Performance Tools (5722-PT1).
1.7.1 What are the PEX Stats collection modes

Here we discuss the PEX Stats collection modes.

The collected data
PEX Stats is the only System i tool that provides both a system wide and individual job view of program information, such as:

- Program name
- CPU usage
- The amount of wall clock time spent in the program (the Inline Elapsed Time)
- The number of disk I/O operations issued by the program
- Counts of PEX events that affect program performance
- Call/return related activity
  - Such as number of times it was called.
  - The number of calls it made.
  - In the hierarchical collection mode.
    - What program called it?
    - What programs it called?
    - How many times each occurred?

**Note:** PEX Trace can provide information about some of the items listed above, but it is not easy for the user. In some cases, there are no PEX Analyzer analysis programs to process the data. Manual analysis can require going through tens of thousands of event trace records to construct a useful view.

The only non-PEX Analyzer command that can print some of the information contained in these records is the i5/OS Performance Tool's Print PEX Report (PRTPEXRPT). The command does not have any analysis or reduction capabilities. We recommend using PEX Trace analysis rather than this command. Handling PEX Trace data takes a lot of your time (literally days in some cases), a lot processing time and temporary disk space, and sometimes does not provide the appropriate solution.

The two PEX Stats collection modes are Flat and Hierarchical. Both of them collect data about programs and the jobs they run in. The Flat mode, which has less collection overhead, does not provide program to program call information. Alternatively, the Hierarchical mode at a higher collection cost, which is determined primarily by program call frequency, identifies, for each called program or MI Complex instruction, the name of the calling program and, when a program calls another program, the name of the called program.

**Flat Mode**
Stats Flat collection data generates one collection record per job per called program or MI Complex instruction. There is no indication that a program was called by one or multiple programs.

**Note:** Unless otherwise specified, this document uses the term “program” to refer to the program/module/procedure combination or MI Complex Instruction.
As mentioned above, the collected data contains, for each program within each job:

- Information about the number of times the program was called
- How many times it called other programs
- The amount of wall clock time spent in a program (from the time it was called until it returned)
- How much CPU time was used in the program
- How many disk I/O operations it caused
  These are identified by synchronous and asynchronous operations, reads and writes, for data base or non-data base objects.
- A total of the same type of information for all the programs it calls or its called programs subsequently call
  For example, program A may call program B, which calls program C, which calls programs E, F, and G. The data for program A includes total Cumulative Elapsed Time, Cumulative CPU time, and Cumulative disk I/O counts used by each program it called and all the programs and MI complex instructions that they called.

**Hierarchical mode**

This mode provides data about each call to a program and puts it into one or more data records (rows) by job-thread. In a hierarchical collection, there is one data record for each unique “calling to called” programs combination within the job-thread. Another way of saying it is “In the PEX Stats Hierarchical mode, the data has one entry per unique call stack for a program within the job.”

For example if, within one job, program B is called by both program A1 and program A2, there will be two different data row entries for each program B call stack in the job. One program B entry represents the PEX Stats information for the call from A1 to B, the other program B entry is for the information for the call from A2 to B. The collected data includes the same information as PEX Stats Flat mode plus:

- There is more information about “Call Level” in 5.2.1, “Call Level, Reference Id, Partial Count Status fields” on page 107.
- A “complete stack” integrity indication. This enables the PEX Stats analysis programs to present data views from which all of the preceding called MI programs can be determined.
- For each program-module-procedure or MI Complex Instruction used within a job, the name of the program-module-procedure that called it and the name(s) of all the programs it called.

### 1.7.2 When is the data collected

When PEX Stats data collection is active and the current job is selected, data is gathered each time any of the following events occur within the job:

- A MI program is called.
  There are also System Licensed Internal Code (SLIC) programs; however, PEX Stats data is not collected for them. SLIC program use can be viewed with the iDoctor's Job Watcher tool.
- A MI program returns to its caller.
- A MI Complex instruction is invoked by an MI program.
- A MI Complex instruction completes.
During collection, if a program or instruction has been called before, the event data is “added” to its existing data. If it has not been encountered before, a new data entry is built to contain the event’s data.

1.7.3 What’s collected for each event

The collected data includes the fully qualified Job name and thread ID as well as the program name, the number of times it was called, and the number of calls it has made to other programs. It also includes the programs inline and cumulative elapsed time (see 1.7.4, “What are inline data and cumulative data” on page 10), CPU time, and disk I/O counts for called programs, modules, or procedures as well as for MI Complex instructions.

Optional special events

There are a number of special event counts that can optionally be collected along with the PEX Stats data. Use them to get additional information about potential problems, such as page faults or other performance degrading activities within a job. More information is in “PEX Event Counts Collection specification” on page 178.

1.7.4 What are inline data and cumulative data

PEX statistics data is collected as one of two types, inline and cumulative.

Inline data

Inline data represents the amount of a resource used and the number of events that occurred within a program.

For example, “Inline CPU time” represents the amount of CPU time used within a program, module, procedure, or MI Complex instruction during the collection. The Inline Synchronous Data Base Reads value represents the number of times this type of read operation occurred while the program was running.

Cumulative data

Cumulative data represents the amount of a resource used or the number of times an event happened within the program plus the amount used within all programs that it calls and within all the programs that those programs call and so on.

For example, the “Cumulative CPU time” value shows the amount of CPU time used by a program and all of the programs, modules, procedures, and MI Complex instructions that it called or caused to be called. For example, if program A called B and B called C:

1. PEX Stats accumulates all of the CPU time used in A, B, and C and puts that into program A’s Cumulative CPU time.
2. Similarly, all of the CPU time used by B and C are summed and put into B’s Cumulative CPU time.
3. The cumulative value for C is the same as its inline amount. In this example, A called B and because B called C, PEX Stats considers that A “caused” C to be called.
If program A is the first program called in a job, it is considered to be running at Call Level 1. When A calls B, B runs at Call Level 2. When B calls C, C runs at Call Level 3. This can continue for many call levels.

In general, a program’s “cumulative” amount of usage is the sum of that program’s usage plus the same values from all programs at a higher numbered (a lower level of control) call level.

**Note:** MI Complex instructions do not call other programs, modules, procedures, or MI Complex instructions. Therefore, their Inline and Cumulative counts are the same. In practice, there are minor deviations from this statement, but it will not affect your Stats data analysis.

### 1.7.5 Why is the Collection Event Count so high

Here we discuss why your Collection Event count is so high.

**Millions and millions of events: is that a concern**

The total number of events that occur during an active PEX collection is maintained and can be viewed. In a PEX Statistics collection, this value represents the total number of calls and returns that occurred in collection jobs. The events that are counted are in the list in 1.7.2, “When is the data collected” on page 9. These counters exist only during PEX Stats collection and are continually being updated during collection and are available after collection ends in the Collection Properties (see “Collection properties” on page 187).

**Note:** You can see an example of the collection’s event count in the “Status” column in Figure 2-36 on page 51.

### 1.8 PEX Stats Quick Start guide

Use the following steps and references to get started quickly with iDoctor and PEX Statistics Flat collection and analysis:

1. Download PEX Analyzer (see 2.1, “Obtaining PEX Analyzer” on page 18). Note the free trial period discussed on the referenced IBM Web site.
2. Install iDoctor and PEX Analyzer (see 2.3, “Installing PEX Analyzer” on page 21).
3. Start iDoctor (see 2.4, “Starting iDoctor” on page 21).
5. Set up to collect Stats Flat data (see 2.7, “Collecting PEX Stats data” on page 28).
7. End collection and save the data (see 2.10, “Ending a collection from iDoctor” on page 49).
8. Run the default Stats Flat analysis (see 2.11, “After collecting: running a Stats Flat analysis” on page 52).
9. Select and view the default reports (see 3.1, “PEX Stats Flat reports” on page 58).
1.9 What tools should I use

In addition to PEX Stats, there are other System i performance collection, reporting, and analysis tools. These include:

**Collection Services (CS)**
Collects comprehensive system wide data sample data on jobs, disks, pools, and communications. Very limited analysis functions. Optionally collects job wait buckets which can be stored along with other collected data into an outfile for later, user written (SQL, Query) analysis. Collection Services is primarily a status gathering tool and along with the Performance Tools provides limited problem determination or analysis.

**Job Watcher (JW)**
Collects system wide or specific job sample data for CPU, object access, multiple types of waits, SQL usage. Has extensive built in data reporting, graphing, and analysis functions.

**Work with System Activity (WSA)**
An interactive Performance Tool's command that samples, displays, and can optionally save system wide job/thread/task CPU and Disk I/O activity. It optionally collects job wait buckets (similar to Job Watcher's wait buckets), which can be stored, along with its normal data into an outfile for follow up analysis.

**Task Switch Trace analysis (PRTTNSRPT)**
The Print Transaction Report provides a limited view of the STRPFRTRC/ENDPFRTRC (Start and End Performance Trace commands) data. No transaction level information; it works well primarily with 5250 or 5250 emulation device I/O.

Using both Job Watcher and PEX STATS takes advantage of the best available tools for finding bottlenecks and scalability/limits to growth situations. Once the analysis tasks are completed and the inhibitors are identified, it is straightforward to evaluate the changes and then proceed to work on the application's workload and capacity scalability.
1.9.1 When to use PEX Stats

Figure 1-1 shows when PEX Stats should be used.

<table>
<thead>
<tr>
<th>When should PEX Stats be used?</th>
<th>CS</th>
<th>JW</th>
<th>PEX Stats</th>
<th>PEX Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What are the jobs doing?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2. Which jobs should I worry about?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3. What are the programs doing?</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Which programs should I worry about?</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Which jobs are using the most CPU?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6. What programs are using the most CPU?</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Why is program X using more CPU or disk IO than it used to use?</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. What jobs are getting the most page faults?</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. What programs are getting the most page faults?</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. What jobs are issuing the most auxiliary storage management requests?</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. What programs are issuing the most auxiliary storage management requests?</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. What programs are improperly structured and need to be changed?</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 13. What needs to be changed: program initiation/termination, use of *
   NEW Activation instead of *CALLER or *NAMED, other? too many Opens/Closes?         |    | x  |           |           |
| 14. What jobs are doing the most disk IO? What Type of IO is it?                               |    | x  |           |           |
| 15. What programs are doing the most disk IO? What Type of IO is it?                          |    | x  |           |           |
| 16. What programs are called the most? What programs are calling them?                        |    |    |           |           |
| 17. Which long running or performance sensitive MI instructions are being used?                |    | x  |           |           |
| 18. What jobs and programs are issuing the MI instructions?                                    |    | x  |           |           |
| 19. What part of the resource usage is attributable to which i5/OS functions?                  |    |    |           |           |

Figure 1-1  When PEX Stats should be used

More information about when to use PEX Stats can be found in 5.4.3, “What are you looking for” on page 116.

Other opportunities for PEX Stats analysis

If the question is “How do I make a job run faster or get more work done in less time?”, PEX Stats provides the means to find where the application can be tuned. It shows “hot spots” of program resource usage and elapsed time costs where the use of specific i5/OS programs or MI Complex instructions indicates secondary hot spots.

One frequent question is “how well will the application scale” (for example, can the application workload grow linearly with increases in processing capacity)? PEX Stats can help you determine that.
It is also very useful to collect concurrent Job Watcher data to see the wait time breakdown into the different wait components and what programs are involved (on the call stack) during the waits. This can help you determine the application’s “scalability factor” for growth. If the jobs are waiting directly for a hardware resource, then more of that resource may help. On the other hand, if the jobs are waiting for internal objects or gates, the waits must be eliminated or the application will not scale with the hardware.

PEX Stats shows what is really going on inside a job in terms of what programs are being used and what are they doing, where the synchronous disk I/O is being used the most, and if data areas or data queues are heavily used. These can be individually quantified for costs and summarized for analysis convenience. If you are collecting Stats Flat data over all jobs, you will see this activity for everything running under i5/OS.

1.9.2 When to use other tools

Figure 1-2 shows you when you should use other tools.

<table>
<thead>
<tr>
<th>What, Why, How Much, Which, is???</th>
<th>CS</th>
<th>JW</th>
<th>PEX Stats</th>
<th>PEX Trace</th>
<th>WSA</th>
<th>TNS</th>
<th>RPT</th>
<th>DB Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Which tool can help determine if performance is good or bad?</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Which tool do I use to see a system view of how my system is running?</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Why is job X using more CPU or disk IO than it used to use?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 How many jobs are currently active?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Is job X multi-threaded?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 How many threads currently exist and how many are consuming CPU?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 Which thread in a multi-threaded job uses the most resources?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 Why is job X taking so long to complete?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Does job X have contention?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 What job is causing object/gate contention?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Which Structured Query Language (SQL) statements are being processed in job X?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 Does the system have a significant number of jobs that are starting and ending?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 Why is the system running slowly?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 What is the impact of page faulting on the system? Are storage pools configured optimally?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 Where is page fault data found?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 Is there any significant save/restore load on the system?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 What is consuming temporary storage?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 Where is my disk space going? Which job or objects are consuming disk storage?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 Which jobs or threads are accessing disk X?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 What is the CPU usage and disk read/write activity generated by the database function?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 Where does my application spend most of the time?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22 Which tools are available to analyze Java performance?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 Which tools are available to analyze Lotus® Domino® performance?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1-2  When other performance tools should be used
1.9.3 Infrastructure comparison with Job Watcher and PEX Trace

The total PEX Stats event count value can get very large. How large depends upon the number of jobs selected for collection, the number of call/returns or MI instructions that occur, and the length of the run. It is normal to see an event count value in the hundreds of millions.

Keep in mind that these values are contained in counters and therefore use a relatively small amount of main storage and disk space. The PEX Stats space needs are a function of the number of jobs, the number of programs that are issuing calls, and the number of MI complex instructions invoked. Most PEX Stats Flat collections have less than ten thousand unique entry records.

Note: For PEX Stats Flat, the maximum number of row entries is the same as the number of records in the collection file QAYPESTATS member.

To compare PEX Stats data collection with Job Watcher’s methodology, Job Watcher uses counters that are continually maintained by System Licensed Internal Code (SLIC) regardless of whether Job Watcher is running or not. One of Job Watcher’s main functions is to periodically harvest the counters and write them to data base files for follow up processing and viewing.

PEX Stats versus PEX Trace overhead

Comparing PEX Stats to PEX Trace, PEX Trace data is inexpensive to gather. However, when PEX Trace collection is shutting down and saving its data, most of the data requires additional processing, so ending PEX Trace collection can be expensive; it depends on the amount of data. Some PEX Trace functions generate a lot of data, while some functions have relatively minimal output data.

In addition, the first analysis of a PEX Trace data collection is orders of magnitude longer and resource consuming than follow-on analysis of the same data. The reason is because the first analysis builds a number of collection related intermediate files (whose names start with SMTR...) in the collection library that are used in subsequent analysis runs to significantly reduce the runtime.

1.10 Stats Data Analysis Examples list

The following lists the example PEX Stats data views in this book. Most of these have associated SQL statements.

1. Figure 3-2, “Sort by Times Called column in descending sequence” on page 64
2. Figure 3-3, “Stats Flat view ordered by Times Called descending” on page 65
3. Figure 3-4, “View sorted by two columns” on page 66
4. Figure 3-10, “MI *CRT... instruction's usage example” on page 73
5. Figure 3-17, “Modified query program selection results: full file Open/Close job candidates” on page 78
6. Figure 3-18, “Analyzing “Times Called values view” on page 80
7. Figure 3-20, “Job’s resource usage summary view ordered by total inline CPU usage” on page 82
8. Figure 3-21, “Cumulative to Inline ratios view” on page 85
9. Figure 5-6, “Stats Hierarchical Analyze Where Used menu” on page 114
Getting started: using Stats Flat

This chapter presents PEX Stats operational information in the sequence in which it is normally used to perform basic functions.

This chapter's topics include:
- Obtaining PEX Analyzer
- Installing PEX Analyzer
- Bringing up iDoctor and PEX Analyzer
- Selecting a PEX data library
- The steps used to collect PEX Stats data
- Generating PEX Stats data analysis reports

Analyzing the PEX Stats data is discussed both in Chapter 3, “Analyzing PEX Stats Flat data” on page 57 and in Chapter 5, “Analyzing PEX Stats Hierarchical data” on page 105.

Additional operational information is discussed in other chapters:
- How to explicitly define the PEX Stats data you want to collect is presented in Appendix A, “Building a PEX Definition for PEX Stats collection” on page 173.
- How to specify the optional PEX Event counts is presented in Appendix B, “PEX Event Counts Collection specification” on page 178.
- How to generate custom table and graph views of PEX Data is presented in Appendix C, “Querying and graphing PEX Statistics data” on page 201.
2.1 Obtaining PEX Analyzer

If you do not already have PEX Analyzer installed on your system, you can obtain a 45 day free trial version by filling out and submitting a trial agreement form located on the Web at: https://www-912.ibm.com/i_dir/idocor.nsf/PATrialAgreement.html

On the trial agreement form, you need to provide:
1. The serial number of your system. To view it, run the command DSPSYSVAL QSRLNBR.
2. Your system’s operating system version, release and modification level (VRM). To view it, run the command DSPPTF.

After the trial agreement form is submitted, you should receive an e-mail reply within 24 hours that contains a temporary access code. The access code can be entered during PEX Analyzer installation or after PEX Analyzer has been installed. It must be entered before PEX Analyzer allows you to start a PEX Analyzer collection or analyze an existing PEX Analyzer collection (one that has been restored from another system).

Note: The system that you install PEX Analyzer on must also have the System i Performance Tools (5922-PT1) installed on it.
As Figure 2-1 shows, all iDoctor tool components including PEX Analyzer can be downloaded from:


Under the “Build and download information” line, select the version you need (V5R4M0, ...). This action brings up the window used to download the tool and view the PTF requirements.

When the trial period expires, you need to purchase a license to continue using it. Use the “Request a price quote” section of the Build and Download window for this process.
Make sure the PTFs shown in the tool’s “Required PTFs” link are applied to your system before installing PEX Analyzer or other iDoctor component. An example of the PTF list is in Figure 2-2.

![Figure 2-2 iDoctor Component PTF list by VRM (V5R3 example)](Image-url)
2.2 Authority requirements

There are different authority requirements for installation and operation of the tools.

- PEX Analyzer installation requires *ALLOBJ and *SECADM authority.
- PEX Analyzer operation requires *SERVICE authority or access to the function group QIBM_SERVICE_TRACE.

Note: Job Watcher operation requires *SERVICE authority or access to the function group QIBM_SERVICE_JOB_WATCHER.

2.3 Installing PEX Analyzer

At the end of the last phase of the download from the IBM.com Web site, a window appears that gives you the option to install the component. Respond to that and follow the prompts.

For more information about installing iDoctor and iDoctor components, see Appendix A, “Installing and uninstalling Job Watcher details”, of the IBM iDoctor iSeries Job Watcher: Advanced Performance Tool, SG24-6474.

2.4 Starting iDoctor

This section shows you how to connect to a system and start iDoctor.

The install process adds a Windows desktop icon that is used to access iDoctor, or you can access the iDoctor for System i link from the Windows Taskbar Start menu. To start iDoctor, double-click the iDoctor desktop icon (see Figure 2-3).

Figure 2-3  iDoctor desktop icon
The initial iDoctor window is My Connections (Figure 2-4). This example’s connections were previously identified as systems on which a user wanted to work with iDoctor tool components.

The first time iDoctor is started after installation, the connections list contains system names defined in iSeries Access for Windows and are automatically added to the iDoctor connections list during its first startup. To add systems to the My Connections menu, right-click anywhere in the view and an Add Connection menu appears. Select Add Connection and follow the prompts.

![iDoctor for iSeries - My Connections](image)

*Figure 2-4 iDoctor’s My Connections window*

When iDoctor starts, the My Connections window entries may or may not be valid. An entry is verified when it is selected and the drop-down menu’s Connect option is selected (Figure 2-5).

### 2.4.1 Connecting to a system

If the system you want to connect to is in the list, right-click it and select Connect from the menu. If it is not in the list, right-click Add Connection... and follow the prompts to add the needed system to the connection’s list.

![iDoctor My Connections menu](image)

*Figure 2-5 iDoctor My Connections menu*
2.4.2 Signing on to a system

After selecting **Connect**, a sign-on window appears (Figure 2-6). Fill in the user ID and password and click **OK**. If you want to sign on to a different system, select **Cancel** to return to the My Connections window to select the one you want.

![Figure 2-6 iDoctor Signon to iSeries menu](image)

The window shown in Figure 2-7 appears after a successful sign-on. Use this window to launch the iDoctor Component(s) you want to use on the system you signed on to.

When iDoctor is started, two server jobs are created. A server job named QZDASOINIT/QUSER is created to handle client SQL requests. Another job QZRCSRVS/QUSER is created to handle the client's remote program calls and remote commands. These jobs are the route used by all user requests from the client to the server. All views are initially built through the QZDASOINIT job and the processing of high priority background work, such as the PEX Stats Hierarchical “where used” and “what used” requests occur in the QZRCSRVS job (unless the user option to submit to batch is used).

If you start another iDoctor session on a different system, another set of jobs are created.

Additional information about the iDoctor jobs is discussed in “iDoctor performance tips” on page 195.

2.5 Starting PEX Analyzer

![Figure 2-7 iDoctor Component's window](image)
The iDoctor Component's example window in Figure 2-7 on page 23 shows there are three iDoctor components available on the system iDoctor is running on. They are Job Watcher, Heap Analyzer, and PEX Analyzer. To use one of them, double-click it or select it and click **Launch**. For example, to use PEX Stats, which is part of PEX Analyzer, select **PEX Analyzer** and either double-click it or click **Launch**.

**Note:** To start another component, go to the Connections window, select the same system and use the iDoctor Components window again. Once you are signed on to a system, it does not ask you to sign on again unless you terminate iDoctor.

To cancel starting a component, click **Close** to return to the My Connections window.

There are a number of possible iDoctor and tool component run time combination options in addition to one system with one component running under one iDoctor session.

1. You can connect iDoctor to multiple systems.
2. You can have multiple iDoctor sessions connected to different systems.
3. You can have multiple iDoctor sessions running on the same system.
4. You can have multiple copies of the same component launched in the same iDoctor session on a system.

**Note:** If you run either one or multiple copies of iDoctor on a system, do not change the JOBQ job scheduling parameters defined during the components installation.

This book does not go into any more detail than this about multiple and concurrent usage, but you may find it useful in some situations.

### 2.5.1 iDoctor components Access codes

Both the iDoctor PEX Analyzer and Job Watcher components require an Access code that is a separate, unique value for each component. The code can be entered either:

- Through iDoctor, when the component is installed.
- Through iDoctor, when the component is launched the first time. Enter the code in the box in the lower left-hand corner of the iDoctor Components menu.
- It can be entered any time after the component is installed by using the server system's command Add Product Access Code (QIDRPA/ADDPRDACS).

The Access code value is unique for each combination of the iDoctor component, the system serial number (system value QSRLNBR), and the i5/OS release level. Access codes can be requested from the iDoctor Web site at:


This URL also has a link you can use to register for a 45 day free trial of PEX Analyzer.
2.6  PEX Analyzer’s initial window

This section discusses the initial PEX Analyzer window. If there is PEX data on the system, it will take a few seconds after you launch it for the initial menu to appear.

2.6.1  PEX Analyzer initial menu list of libraries

Figure 2-8 shows the initial menu list of the PEX Analyzer's libraries.

The PEX Analyzer startup window has an “Explorer-like” view of all libraries that contain PEX Data. A library can contain any or all of the three types of PEX data: Statistics, Trace, or Profile. This book discusses only PEX Statistics data.

Clicking the + sign next to PEX Analyzer on the left side of the window expands the sign into a scrollable list of all PEX data libraries. You can scroll down the right hand side of this window to position to the library you want to work with without expanding the left hand side.

For example, in Figure 2-9, assume you want to work with the PEX data in library “RTCALLTEST”. You know that it has PEX data in it, otherwise it would not be in the list. The user provided “Description” (which is the library’s descriptive text) column may tell you more about what is in the library.
When you double-click the library name on the right-hand side, the menu view changes to show all of the libraries PEX Collections (Figure 2-10). At the same time, the left hand side of the window's view changes to show all the PEX data libraries.

Once you are at this point in starting PEX Analyzer, you normally take one of two paths:
1. You can view a library's PEX data collections and their related analysis reports. This is discussed briefly in 2.6.2, “PEX data collections in a library” on page 26.
2. You can initiate a PEX Stats data collection, as discussed in 2.7, “Collecting PEX Stats data” on page 28.

2.6.2 PEX data collections in a library

Any library can contain one or more PEX data collections. The example in Figure 2-10 shows four collections in the selected library. Double-clicking a collection brings up a window showing the analyses that have been run on the data (Figure 2-11 on page 27).

![Figure 2-10  PEX data collections in a library](image)

2.6.3 Viewing the PEX Analyzer reports

Figure 2-11 on page 27 shows that a PEX Stats analysis named “Stats flat analysis for all jobs/threads” was run on the first collection (Calltestsf). This is shown by the Analysis Name column entry and the Status column’s complete value.

If no analyses are shown, either the data has not been analyzed or the analysis was run and then deleted at a later time. Deleting an Analysis and other library and analysis related utility functions are discussed in “Library Submenu Options” on page 184.

Running an analysis is a user initiated operation. To run an analysis, select the collection and right-click it to get the analyzer menu. This is discussed in more detail in Figure 2-38 on page 53.
Each collection that has been analyzed can have one or more Analysis Name entries and each analysis entry has one or more report folders. Select the Analysis Name list entry (Figure 2-11) to view the Collection's Analysis Report folder list (Figure 2-12). Double-click Report folder **Analysis reports** to see the contents of the Report folder.

Double-clicking the Report folder column labeled **Analysis reports** brings up the Report description entry’s window (Figure 2-13 on page 28). This shows the name of the two default PEX Stats Flat data reports. These were generated by the PEX Stats Flat analysis:

- Summarized CPU and I/O by Job/Pgm/MI Instr
- Summarized CPU and I/O by Pgm/MI Inst

The highest level summary is the Summarized CPU and I/O by Pgm/MI Inst. It contains one line per unique program/module/procedure or MI Complex instruction. It is often the best starting point for working with the Stats Flat data. The report data comes from the analysis output file named G_STATSFPG (Stats Flat by program), as shown in the Filename column.

The report labeled Summarized CPU by...Job/Pgm/MI Instr has one line *per job* per unique program/module/procedure or MI Complex instruction. The report data comes from the analysis output file named G_STATSFJB (Stats Flat by job and program), as shown in the Filename column. Additional information about this file is in “PEX Stats data analysis output files field names: V5R3 / V5R4” on page 244.
These reports are discussed in Figure 3-1 on page 58. That view is an example of the Summarized CPU and I/O by Pgm/MI Inst view and discusses some of the data field’s content and meaning.

Figure 3-12 on page 74 is an example and discussion of the Summarized CPU and I/O by Job/Pgm/MI Inst view.

### 2.7 Collecting PEX Stats data

PEX data collection requires a definition about what to collect plus control information. The “what to collect” information and some of the control information is provided in a PEX Definition object.

Prior to collecting PEX data, you define what you want to collect by:

- Selecting a PEX Analyzer supplied PEX Definition
- Building your own PEX Definition object

iDoctor provides an interface to select an existing PEX Definition or to build or modify an existing PEX Definition.

Each PEX Definition, which can be built by iDoctor’s PEX Analyzer GUI or by the QSYS/ADDPEXDFN command, is stored as a separate member of file QAPEXDFN in library QUSR禄S.

**Note:** Performance Explorer is part of i5/OS. The PEX commands that need a PEX Definition always get it from this file and library. There is no command to override or redirect where the PEX commands look for it.

You can work with the PEX Definitions using iDoctor’s PEX Analyzer GUI. If no PEX Definitions have been built on your system, the QAPEXDFN file does not exist. The file is created, when needed, by the PEX support. You do not have to manage it.
2.7.1 What is in a PEX Definition

A PEX Definition, which guides the PEX Data Collection functions, contains the following information:

1. It has a unique name and optional description.
2. It defines the type of data to collect. This book discusses only PEX Stats Flat and PEX Stats Hierarchal data.
3. It defines to PEX collection functions what jobs to collect Stats data for. The collection scope can be for all jobs on the system, a set of jobs, or one job.
4. It defines what SLIC tasks to include information about in the collection properties.

The collection properties contain task level CPU and disk I/O usage data. This is also collected for jobs and threads. There is no PEX Stats data for SLIC tasks because their programs do not use the MI Call/Return interface to invoke other programs.
5. It defines optional event collection (this topic is covered in “PEX Event Counts Collection specification” on page 178 and in “PEX event definitions” on page 190).

Starting a PEX collection requires, in addition to the PEX Definition, the following additional parameter definitions:

1. The Collection name.
2. The collection library name where the collection data files are to be stored.
3. When the collection is to start (right now or sometime later).
4. Implicitly, when the collection is to end; specifically, how long to collect for.
5. How the collected data is to be stored after the collection finishes. There are two options:
   a. Save the data in a set of PEX defined and built data base files. These can be analyzed on the collection machine or can be sent to another system using iDoctor library utilities (discussed in “Library Submenu Options” on page 184).
   b. Save the data in a Management Collection (*MGTCOL) object that contains the data in a compressed form. Use this option for a collection that is to be sent to another system for analysis to minimize the transmission overhead. The data can be extracted with the CRTPEXDTA command.

PEX data is not stored until after the collection has finished. This differs from Job Watcher and Collection Services, which can store the data into data base files during collection.

PEX data for collections that have not completed is not retained across an IPL. Any suspended sessions are erased during an IPL.

You can run concurrent collections. See 2.13, “Cautions for analysis” on page 55 for more information about multiple concurrent collection sessions.
2.7.2 Library selection

The first step in starting a data collection is to tell iDoctor that you want to create a collection. There are two ways to do this:

- In PEX Analyzer's initial window, right-click PEX Analyzer and select Create PEX Collection... from the drop-down menu (Figure 2-14)
- or
- Scroll down the Library Name list in the right hand window (or the expanded left hand window not shown here), select and right-click the library you want to use, and select Create PEX Collection... from the drop-down menu, as shown in Figure 2-15.

If you choose the first step, then later you specify the library name. If you did the second step, the library is selected implicitly.

Figure 2-14  Create PEX Collection menu from the PEX Analyzer window

Figure 2-15  Create PEX Collection menu from the Library Name window

The first option (selecting from the PEX Analyzer entry) requires filling in the collection library name on the Collection Option's menu (Figure 2-19 on page 34).
If the library does not exist, iDoctor offers to create it for you after it is named on the options menu.

The second library selection option fills in the menu’s library name field with the name of the selected library. It can be changed on a subsequent menu.

### 2.7.3 Basic or Advanced path selection

You can select one of two paths to follow through the menus to set up and run a PEX Stats collection. Use the Basic path unless you want to specify more than the basic path provides, such as Event Counter specification and collection.

To create your own PEX Definition, see Appendix A, “Building a PEX Definition for PEX Stats collection” on page 173.

Figure 2-16 shows the Welcome window.

![Figure 2-16](image)

**Figure 2-16** Starting a PEX Collection Welcome window

Use this window to select which path to take to start collecting PEX Stats data.

- The “Basic” path leads you through a set of questions to help you decide what to collect and builds a PEX Definition to use. On the Option's window, you can override the PEX Definition specification and select some other, pre-existing user defined definition. If you are going to do that, it is better to start with the “Advanced” path discussed in 2.8, “Advanced Path PEX Options specification” on page 42.

- The “Advanced” path takes you to the Options window, which follows the last window of the basic path.

If you want to use an existing, user defined PEX Definition, set the desired mode to **Advanced** instead of **Basic** and select **Next**. That takes you to the window shown in Figure 2-27 on page 42.
2.7.4 Select FLAT or HIERARCHICAL collection type

There are a number of problem types shown in this window (Figure 2-17). For PEX Stats, either Flat or Hierarchical collection, select the first (default) option **Excessive CPU utilization**.

The other options are for PEX Trace or Profile and are not discussed in this book.

*Figure 2-17  Create PEX Collection - basic path - Problem Type Selection-1*
In the next window (Figure 2-18), there are two options to select from that apply to PEX Stats collection:

1. To select Stats Flat data collection, select the default **CPU statistics by program/procedure/method**.
2. To select Stats Hierarchical data collection, select **CPU statistics by program/procedure/method with call flow**.

![Figure 2-18 Create PEX Collection - basic path - Problem Type Selection-2](image)
Selecting **Next** in this window and in the following one, which is an information only window and is not shown, takes you to the Collection Wizard’s Options window shown in Figure 2-19.

### 2.7.5 PEX Stats Collection Options specification: Basic Path

![Create PEX Collection - basic path Options window](image)

**Figure 2-19**  Create PEX Collection - basic path Options window

**Setting the parameters in this menu**

**Collection name**  
This is the member name that is added to each of the collection’s PEX data base files in the collection library when the collection starts. The collection data base file names all start with QAYPE.

**Library**  
The name of a library where the collection data base files are stored. The library does not have to exist at this point in the process.

**Description**  
An optional line of text that describes the collection.

**Duration**  
The collection run time, in minutes. iDoctor provides a menu you can use to explicitly stop the collection before its elapsed time expires.

If there is a doubt as to how long to collect, set the time limit longer than the test rather than shorter. The reason is that if the collection stops before the test case finishes, the PEX Stats data information is not complete and there is a high probability of misinterpreting the data.

More information about collection duration is in “Duration value: How long to collect” on page 43.

Use the default value for the remaining parameters.
Notes about the Options window:

1. Basic Path: One difference between the basic and advanced paths is that the basic path selects a PEX Analyzer supplied PEX Definition named STATSFLAT. This definition is modified with information for this collection, and a new PEX Definition is created with the same name as the PEX Collection.

2. Basic Path: On the Options window, the Details button is not available to show the definition's properties.

3. Basic Path: As mentioned above, the collection name also becomes the name of a new PEX Definition, implicitly created by iDoctor, and used to control the collection. It is created and stored in the QUSRSYS/QAPEXDFN file with all other PEX Definitions.

   Be careful when selecting the collection name. It is possible that you could delete an existing PEX Definition with a matching name that belongs to you or to some other user.

   It is also possible that you could inadvertently overlay an existing collection if you define one with the same name as one that already exists.

   You have a lot of authority while running iDoctor tools, so be sure to be careful when using it.

4. If you are going to run multiple simultaneous or overlapping collections, you must define a separate library (and collection name if you are using the Basic path) for each collection.
   - iDoctor does not check for this type of run configuration. Concurrent collections into the same library cannot be controlled by iDoctor and PEX Analyzer (even if you are using PEX Analyzer command-line commands, which are not discussed in this book) and you will experience severe collection data integrity problems that you will not recognize or be able to correct. There is no difficulty if a different library is used for each of the concurrent collections.

   To reiterate, everything will work fine if you use different library and collection names in this case.

   - It is not a good idea to use a common library such as QPEXDATA as the default output library (for reasons of PEX Definition and PEX data collection integrity).

   - It is best to have each user pre-build their PEX Definitions and collect into their own, uniquely named library to ensure data integrity.

   For more information about this and other operational concerns, see 2.13, “Cautions for analysis” on page 55.

Fill in the remaining options and select Next to proceed to the PEX Collection Wizard - Job/Task Options window in 2.7.6, “Job/Task selection” on page 36.
Create library prompt
Figure 2-20 shows the create library prompt.

![Create library prompt](image)

*Figure 2-20  PEX Collection Options: create library prompt*

If the destination library does not exist after clicking **Next** on the Options window, iDoctor offers to create it for you. The options are:

- Yes to create the library and proceed to the next menu
- No to return to the Options menu without creating the new library

### 2.7.6 Job/Task selection

This section discusses the iDoctor steps used to specify which jobs are to be enabled for PEX collection.

All jobs in the system can be selected or the selection can be restricted to specific jobs, user IDs, or job number. If you already know which jobs you want to enable, perhaps based on previous collection work or user input, you can gain a collection performance advantage by limiting the selection to just those jobs. That reduces the collection CPU overhead. Reducing space usage is not a concern with Stats data.

There are iDoctor menus to use to specify job selection. The same set of menus are used when building a PEX Definition as well as in other PEX Analyzer component's collection setup functions.

There are two ways to get to the Job/Task Options menu:

1. While starting a collection, you select a PEX Definition from the Options window Collection Options frame "Definition type: PEX-Analyzer supplied" list on the iDoctor window shown in Figure 2-19 on page 34.
2. Creating a PEX Definition with PEX Analyzer.

**Note:** There are no job selection options presented when you use a existing User Defined PEX Definition. The job selection criteria were defined and stored in it when it was created and cannot be changed during collection startup.
Figure 2-21 shows the PEX Collection job options.

![Figure 2-21 PEX Collection Job/Task Options](image)

This window's content is context sensitive depending what the user is doing. For example, Current job appears only in certain instances.

**Type of job selection to use**
Select **All jobs**, **Selected jobs**, or **Current job**.

**All jobs**
Causes the remaining job selection windows to be skipped and processing to proceed to the final menus discussed in 2.8.1, “Ready to go: starting collection” on page 46.

The jobs may or may not have existed at the beginning or end of collection. The data is collected for all jobs that existed during the collection period.

All jobs is available only for Stats Flat collection. Stats Hierarchical requires job selection specification.

**Selected jobs**
Uses the following windows to select which jobs you want to enable for Stats data collection.

If you are going to collect Stats Hierarchical data, you must explicitly select which jobs you want to undergo collection. Generic job selection (QPADV*V, for example) can be used.

The type of selection can be either the full job or user name or, alternatively, generic selection. Generic selection may be used for both the job name and user ID.

**Current job**
The job that starts the collection is the one that undergoes collection. This could mean an interactive or batch job's CL program issues the STRPEX and ENDPEX commands as well as doing the work that is being measured. See the discussion in “The fastest path in and out of collection: STRPEX and ENDPEX” on page 195 for more information.
**Job Selection value specification menus**

When the job selection window (Figure 2-22) is initially shown, it is immediately overlaid with the Add Jobs window (Figure 2-23).

When the Add Jobs window is closed or it is dragged to a different location, the Job Selection window can be seen and contains any job selection criteria that has been specified.

![Figure 2-22 Job Selection initial window](image)

**Consideration:** There can be up to eight (8) job selection specification entries. However, this does not mean that there is an eight job limit on the number of jobs undergoing collection. It means that there can be up to eight job selection specification entries. The use of generic job selection enables any number of jobs to undergo collection. The number of jobs that may be selected to undergo collection may also depend on how the user’s job naming conventions fit in with the selection specifications.

Because generic selection can cause many jobs to undergo collection, you may collect data on jobs that you did not want.

![Figure 2-23 Job Selection “Add Jobs” initial window: initial view](image)
For example, if you want to view all jobs currently running on the system whose name starts with “QPADEV*”, enter that value into the Add Jobs window’s Job Information option’s box Name field and click either Add or Refresh.

1. Clicking Refresh updates the Add Jobs menu’s Active jobs matching job information list to show all jobs that match the Job Information frame’s criteria (see Figure 2-24).

   This is a list of possible job selection candidates. At this point, you can select one or more specific jobs from the list, change the criteria and refresh the list, or cancel.

   Note the Refresh option does not select jobs for collection; its purpose is to display candidates. Setting the selection criteria occurs only when the entries are put into the Add Jobs window (shown in Figure 2-26 on page 41).

2. Clicking Add adds the selected Job Information criteria into the Job Selection window’s selected Jobs to collect list. Either generic or full job ID selection specification or both can be added into the Job Selection list.

   See Figure 2-25 on page 40 for an example.

![Figure 2-24 Job Selection Add Jobs selection example view](image)

This Add Jobs window shows the result of entering QPADEV* in the Job Information Name field and clicking Refresh. All active jobs on the system you are connected to whose names satisfy the selection criteria appear in the list.

The list shows jobs that are currently active in the system. If you specify generic job selection, all jobs that are active during the collection and satisfy the selection criteria are enabled for collection. Stating it another way, this means all jobs that met the selection criteria and were running at collection start or started running after collection began are enabled for collection.
Prior to using the Add Selected option, you can select any combination of Job Name, User ID, Job Number, and Current user. For example, Figure 2-25 shows multiple jobs selected to be added to the Job Selection window.

Select multiple rows using the standard Windows selection methodology by holding down the Ctrl key and clicking each desired row. After all the rows are selected, click Add Selected to add them to the Job Selection window’s jobs to collect list (Figure 2-26 on page 41).

You can use the Add Jobs window to make different job name/user ID/current user combinations and add them to the Job Selection. When you are finished with job selection, click Close on Add Jobs. The Job Selection window contains all of the specified job selection criteria.

Figure 2-25  Job Selection “Add Jobs” job selection example
After clicking **Add Selected** on Add Jobs (Figure 2-25 on page 40), the Job Selection window will look like the list shown in Figure 2-26.

![Figure 2-26 Job Selection example view](image)

Remember, there can be up to eight entries in the Job Selection window’s list of Jobs to collect.

The Job Selection window button’s can be used to remove an entry, to go back to the Add Jobs menu to add in other job selection criteria, or go back to the Options menu. You can:

- Select one or more rows and click **Remove** to remove selected entries.
- Select **Add Jobs** and go back to the Add Jobs window (Figure 2-25 on page 40).
- Select **Back** and go back to the Job Task Options window (Figure 2-21 on page 37).
- Select **Next**, which adds all the jobs in the list to the collection and takes you to the Collection Summary window (see 2.8.1, “Ready to go: starting collection” on page 46 and Figure 2-30 on page 46).

**Task selection**

PEX Stats does not use named task selection. Stats data is not collected for SLIC tasks. This is because SLIC programs do not use the same call/return functions that MI programs use and PEX collection records.

There are two task selection options on the Job/Task Options window (shown in Figure 2-21 on page 37) when specifying Stats Task Selection.

- **All tasks**: If you want task selection, this is the option to use.
- **None**: This option shows some SLIC task related information.
What does Task Selection do for me

Task Selection options affect the collection's data shown when you use the Collection Properties Collection task tab view (Figure B-6 on page 188).

- **All tasks**: The collection properties show the resource usage data for every SLIC task active during the collection.

  Use this setting to see what the task run time data looks like. Sometimes it is useful to see what tasks are running, especially those that directly support application processing, such as the:
  - Data Base Level 3 Server Tasks (DBL3....)
  - Journaling and SMAPP tasks (JO....)
  - Error Logging tasks (ERR...)
  - Storage Management High Priority Page Out tasks(SMPO001)

- **None**: The collection properties show only the CFINTnn task information. This is not useful data in PEX Stats analysis.

### 2.8 Advanced Path PEX Options specification

As you are creating a PEX Definition or setting up a Stats collection run, selecting the **Advanced Path** option (Figure 2-16 on page 31) causes control to go to the Options window (Figure 2-27). Some collection parameters are defined here before moving on.

![PEX Collection Wizard - Options - Rchassq1](image.png)

1. Click here to open the list of PEX Definitions

2. After selecting a PEX Definition, click “Details” to see its properties

*Figure 2-27  Selecting an existing user-defined PEX Definition*
To select the collection’s PEX Definition:

1. Select the **User-Defined** option button.
2. Select the **Definition list header** to open a list of all the PEX Definitions.

**Note:** The list of PEX Definition names is a list of the member names in the QUSRYSYS/QAPEXDFN file.

3. Select the PEX Definition you want (this example uses “ALLSTATSF”, as shown in Figure 2-28 on page 44).
4. Select **Details** to see the selected definition’s properties summary.
5. Fill in the remaining menu fields:
   - **Collection Name**: Whatever you want to name the collection.
   - **Library**: The name of the library where the collection is to be stored.
   - **Description**: An optional text description of the collection you are creating.
   - **Duration**: Specify the length of time, in minutes, to run the collection.

**Duration value: How long to collect**

A PEX Stats collection contains program CPU time, elapsed time, disk I/O counts, and program to program call activity. You should collect enough information to accurately characterize the program’s behavior. How long you run the collection depends on multiple factors:

- Are you collecting over the whole system or a subset of the jobs?
  - If your objective is to get an overview of program activity, 10 to 30 minutes on an active system usually provides a good sample of what the programs and jobs on the system are doing.

    This assumes that you are collecting while the system is doing normal or higher than usual workload. Collecting over a system that is idling does not tell you anything.

  - Collecting for an extended period of time is not a good practice, because the Stats collection processing can possibly cause up to 10 to 20 percent performance degradation in throughput and response time. On a very busy system, there possibly will be performance degradation that the customer will experience so you want to minimize the disruption.

  - Sometimes, in very busy systems, a one to five minute collection will give you useful data, but that short of a collection probably will not provide a valid representative view of system activity.

  - Sometimes collecting for longer durations is necessary to see all of the programs in a critical path (such as overnight batch). Some customers collect Stats Hierarchical data for their overnight process on a regular basis.
What type of data to collect
What are you collecting, Flat or Hierarchical data?

- If you are profiling the system to see what functions are used and what their resource usage is, run a Flat collection.
- If you looking at the overall performance of specific jobs, a Flat collection will work.
- If you want to see the performance of small set (ten to twenty) of specific jobs or need to see the program to program flow (again, of a small set of jobs), collect Hierarchical data.

Cautions

- Never collect Stats Hierarchical over the total system.
- If you are collecting hierarchical data, make doubly sure that you limit the number of jobs enabled for concurrent collection to 10 to 20 jobs, even on big systems.
- Be aware that there could be a lot of active jobs whose names match a generic selection definition. It is possible that this type of specification can raise the number of collected jobs well above the suggested number of concurrent job collection limit.

Scheduling the Start time
To use iDoctor’s Collection Scheduling functions, click Schedule Start Time and follow the prompts.

![Figure 2-28  PEX Collection Wizard - Options window - other settings](image)
Clicking the Collection Options box **Details** button shows you the selected PEX Definition's Properties (Figure 2-29). Clicking **OK** on that window returns to the Options window.

![Selected PEX Definition's detail properties](image)

**Figure 2-29  Selected PEX Definition's detail properties**

This window shows you the PEX Definition's properties. Clicking **OK** takes you back to the Options window where you can select a different PEX Definition, enter other options, or proceed to the next setup options window.
2.8.1 Ready to go: starting collection

Note that in the Advanced path where a *user defined PEX Definition* is selected, there is no job selection definition capability. In this case, that information must be put into the selected PEX definition when it was built.

However, if you selected a *PEX Analyzer supplied* PEX Definition, in either Basic or Advanced mode, iDoctor takes you to the Job/Task Selection window next. After completing job selection, the next window is the Summary menu.

At the Summary window (Figure 2-30), you can do any one of the following:

- Start the collection by selecting Finish.
- Submit it if you used the iDoctor Scheduler.
- Go back and redefine parameters.
- Cancel the request.

![Figure 2-30  Create Collection Summary and final window](image)

Note the string of data following the Remote Command String. This is an iDoctor STRPACOL (Start a PEX Analyzer Collection) command. This command is not discussed in this book other than to say it does not support multiple concurrent collections or some of the ADDPEXDFN command options. Click **Finish** to start collecting.
2.9 Monitoring active collections

Soon after the collection starts, the window in Figure 2-31 appears that shows the PEX Libraries.

![Figure 2-31  PEX Analyzer window of PEX data libraries](image)

If iDoctor was given a new library to start the data collection in, the library was created (see 2.7.5, “PEX Stats Collection Options specification: Basic Path” on page 34).

For the new library name to appear in the list of libraries, the window has to be refreshed. You can explicitly refresh it, which adds the new library to the list of libraries, by doing one of the following:

- Click the iDoctor “circular arrow” icon to refresh the window.
- Move the cursor into the list of library's window on the right hand side and press F5.
The result is similar to the window in Figure 2-32.

2.9.1 Collection run status and control

The next window (Figure 2-32) shows the effect of the refresh. In this example, the new library “my_library” was added to the list of PEX Analyzer libraries.

Select the new library to see the collection status.

![Refreshed Library Name view showing the new collection library](image)

Figure 2-32   Refreshed Library Name view showing the new collection library

Run status

When the new collection library is selected, the collection’s status information appears (Figure 2-33 on page 49).

In this example, the Status column indicates “in progress” and its Events count value is the number of events that have been collected. As the enabled jobs perform work, the event count increases. To update the Events counts value, press F5 with the cursor in the view or click the refresh icon (the iDoctor clockwise arrow icon).

You can also monitor the count from a i5/OS command line by invoking ENDPEX with no parameters. In its view, you can repeatedly refresh (F5) the active collection’s event counts. Be careful; do not inadvertently enter an option value and press enter in the ENDPEX list of active collections.

The Type column In Figure 2-33 on page 49 contains information about the collection’s PEX Definition. If its prefix is User-defined, it is followed by the collection’s PEX Definition name. If its prefix is PEX Analyzer-supplied, it is followed by the name of the PEX Analyzer definition used to create the real PEX Definition used and the definition’s name is the same as the collection name.
2.10 Ending a collection from iDoctor

Figure 2-33 shows the collection, status, and control sub-menus.

![Collection status and control sub-menus](image)

Right-click an In Progress collection to bring up a menu from which you can:

1. **End Collection**: Selecting this brings up an options menu to select from.
   a. **Create DB files**: creates the collection’s data files (members) that are input to the PEX Stats Analysis programs.
   b. **Create a single object from the collection data**: Use this option when you intend to transport the data to another system or archive it locally without analyzing it immediately. This option creates a Management Collection (*MGTCOL*) object with the same name as the collection that contains the collection data in an internal form.

   See 2.10.1, “MGTCOL object processing” on page 50 for a discussion of the MGTCOL object processing options.

   c. **Delete**: Stop data collection data and discard the data.
   d. **Stop**: Stop data collection but do not discard the data.
      i. The collection cannot be restarted.
      ii. You can right-click the collection and select from the menu to create the DB files from the data.

2. **Restart**: Stop the current collection, throw away the current data, and restart collection with the same parameters used when it started.

3. **Properties**: Shows the Collection Properties menu. For more information, see “Collection properties” on page 187.
2.10.1 MGTCOL object processing

When the data is saved as a MGTCOL object, the list of collections could have an entry that looks like the “Test1” collection in Figure 2-34.

![MGTCOL object menu options](image)

In this example, right-clicking collection **Test1** brings up a submenu with the following options:

- Convert into PEX collection...: Convert the MGTCOL data into a set of PEX Collection data files. See Figure 2-35 on page 51. These files can be processed with PEX Analyzer.

  This procedure does not delete the MGTCOL object; do that manually.

- Copy: Copy the MGTCOL object to another library on the current system.

- Delete: Delete the MGTCOL object. Do this after converting the data.

  **Note:** There is no way to recreate a MGTCOL object from the PEX data files.

- Transfer to...: Save the collection’s MGTCOL object, and transmit and restore it to another system.

- Properties: View the MGTCOL object's properties.
Chapter 2. Getting started: using Stats Flat

Figure 2-35 Converting a MGTCOL object to PEX Analysis input files

The Number of threads... option is not necessary for Converting Stats collections because they do not have enough data to benefit from using multiple threads.

Selecting **OK** submits the QSYS/CRTPEXDTA command to batch. This command is part of i5/OS PEX.

- The command converts the MGTCOL object's data into PEX Stats Collection data files in the specified library.
- The file’s member name is the same as the Collection name from this window.

Figure 2-36 Collection stopped window

The collection Status column shows the collection run status and the number of events. The Type column shows if a collection uses a user defined or PEX supplied PEX Definition.

The number of events can be very high, in the multiple millions. These are counts of calls and returns. The amount of data stored is relatively low. For more information, see 1.7.5, “Why is the Collection Event Count so high” on page 11.
2.10.2 iDoctor utilities

When you use some of the iDoctor utilities, such as MGTCOL object conversion, or delete PEX data, the activity is logged in the iDoctor Remote Command Status window. This window can be selected for viewing from iDoctor's main menu’s Windows command.

![iDoctor Remote Command Status window](image)

The window in Figure 2-37 shows the command status that resulted from running a CRTPEXDTA command (row 1) and the deletion of a set of files (row 2) and of a MGTCOL object from a library (row 3).

2.11 After collecting: running a Stats Flat analysis

This section discusses the PEX Analysis wizard's activities and submitting collections for analysis.

**Note:** PEX Stats data analysis works only with data from the same i5/OS VRM that it (PEX Stats Analysis) is running on. It does not analyze data from previous releases.

You can view PEX Analysis reports from data that goes back two releases or later and use user defined queries on them.

You can process the internal QAYPE... files from any release with your own queries. These files are not covered in this book.

2.11.1 Analysis wizard and submitting to batch

Right-clicking a collection from a library’s Collection window brings up the Analyze Data window. From there you can either:

- Run a default analysis.
- Start the analysis wizard. This option takes you through a set of menus to select specific collection jobs to run the analysis over. More information about multiple job analysis is in 4.4.1, “Multiple job analysis selection” on page 100.

“Running the default” is the recommended option to do the initial analysis for Stats Flat data.
Figure 2-38 shows the Submit Collection Data Analysis job menus.

After the job is submitted, a confirmation window follows showing that the analysis job was submitted. The default is to submit it to the iDoctor batch job queue. Do not use any other job queue for PEX Stats analysis jobs.

2.11.2 After the analysis: viewing the reports

Figure 2-39 shows the PEX Stats Flat analysis Report descriptions.

To see the analysis results, select a Report Description entry. The path is Library → Collection → Analysis files → Report Description → Report.

When you are first working with Stats analysis, it is easiest to start with the report labeled Summarized by ... Pgm ..., where there is one line of data per Called program or MI Complex instruction. When you have found program or instruction entries that you want to learn more about, use the report that contains both job and program data. The Stats Flat report by job/pgm... contains one line per Called Program or MI Complex instruction per job.
More information about the Pgm file’s window is in Figure 3-1 on page 58; the Job file’s window is shown in Figure 3-12 on page 74.

2.12 Cautions for data collection

When running multiple concurrent PEX collections, you must put each collection into its own separate library to avoid data integrity problems when the collection is ending. In addition, ensure that each collection name is unique.

Ignoring these cautions results in a high probability of unpredictable and confusing results (if any at all).

**Note:** Multiple concurrent session are OK if you follow the rules, but you *must* follow the rules.

The i5/OS PEX (*Performance Explorer*, not *PEX Analyzer*) commands are mentioned briefly in this book in “The fastest path in and out of collection: STRPEX and ENDPEX” on page 195.

Do not use these commands without understanding their interaction with PEX Analyzer.

Observe the following two rules (as well as the others stated earlier for multiple concurrent collections) and your data will be in good shape:

1. Do *not* start a collection with iDoctor's GUI and end it with ENDPEX.
2. Do *not* start a collection with STRPEX and end it with iDoctor's GUI.

These are incompatible combinations and if they are used, you will have a hard time piecing the data together with the PEX data analysis functions.

The same rules apply to multiple concurrent data analyses, even if you have multiple iDoctor sessions on the same system.

**Additional information**

There is only one additional consideration that applies if you decide to run multiple concurrent PEX TRACE collections: you *must* ensure that the CPU sampling interval on the CPU event (called Performance Measurement Counter Overflow (PMCO)) has the same value specified for *all* concurrently active PEX TRACE collections.

There are no such concerns for PEX STATS or Job Watcher, as the PMCO event is not used by those types of data collection.
2.13 Cautions for analysis

PEX Analyzer was originally built to support a single user. iDoctor, built much later, supports multiple sessions and multiple users on the same system simultaneously.

PEX Analyzer has had changes to adapt to the single versus multiple user scenario. For the most part, it works properly, but there are still some things that you need to be aware of:

1. Do not change the number of concurrent analysis jobs in the PEX Analyzer related JOBQ that was created when the component was installed.

2. All PEX Analyzer submitted batch jobs must run serially, one at a time, or once again you will encounter unpredictable and confusing results and possibly the only way out is to delete the data and rerun the collection.

2.13.1 Keeping track of your data

It may make sense to use a naming convention for collections and libraries prefixed with the following:

<table>
<thead>
<tr>
<th>S</th>
<th>PEX STATS</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>Job Watcher</td>
</tr>
<tr>
<td>T</td>
<td>PEX Trace</td>
</tr>
</tbody>
</table>

For the collection name, you may wish to use date time 17021130 so that you can readily identify the content without having to always refer back to the collection properties shown in “Collection properties” on page 187.
Analyzing PEX Stats Flat data
3.1 PEX Stats Flat reports

This section discusses the Stats Flat default analysis reports. It provides both an introduction to Stats Flat data analysis and discusses additional analysis options that are available by modifying the data views to help achieve your analysis objectives.

3.2 PEX Stats Flat Program default report view

Figure 3-1 shows the default PEX Stats Data by Program window. Each row shows the resource usage, time, and count data for a called program/module/procedure combination or MI Complex Instruction. The window has as many rows as there are unique program / module / procedure or MI Complex instructions in the collection data.

Note: Note that the view of some column's data values is truncated, as indicated by a “>” character at the right hand side of the column's data.

To see the complete value, expand the column width by placing the cursor on the column header's right hand boundary and drag it to the right until the full data is shown.

Some of the data columns in the default PEX Stats Flat view are not shown here due to space limitations. They are discussed in “Other fields” on page 61.

Each row's data columns describe (Figure 3-1):

- Library name: The name of the library that the called program was located in. For MI Complex instructions this field is blank. Usually the i5/OS system library names start with “Q”.
- Program name: The name of the MI program that this data applies to.

To see an i5/OS program's short description move the cursor over the row's Program Name value. A short “flyover” description of the program may appear.
See “i5/OS programs” on page 236 for a discussion of some of the more likely encountered operating system programs (from a performance perspective).

Every System i MI program is built to conform to either the OPM or ILE program model. All programs that you want to undergo collection must be properly enabled for collection. By default, all OPM programs are always enabled; ILE program’s default creation parameter enables them for Procedure Entry Point (*PEP) collection.

However, in the case of an ILE Service Program, this default creation parameter setting does not enable it for collection. The reason is that ILE Service Programs do not have a PEP. They need to be enabled at their entry point (for example, use ENTRYEXIT to enable them).

To set the service programs correctly (through V5R4) the Create XXX Module (CRTxxxMOD, where xxx is RPG, CBL, and CL) command’s ENBPFRCOL parameter must be set to ENTRYEXIT or FULL to enable ILE service programs for collection. (Using FULL is more than needed; ENTRYEXIT is recommended).

This discussion also applies to ILE programs containing multiple modules and procedures, which is becoming more commonplace in advanced programming shops. The creation default of *PEP does not show procedure level statistics. A good practice is to compile with *ENTRYEXIT for true ILE programs (not single bound modules). Also, the CRTSQLxxx command does not have an ENBPFRCOL option. A CHGPGM or CHGSRVPGM is required to enable PEX at the procedural level.

**Note:** These same collection enablement rules apply when the i5/OS STRTRC/ENDTRC commands or PEX Trace’s Call/Return Trace entries (MENTRY and MIEXIT event specification) are used to capture the program’s activity. If the ILE service program is not set up with ENTRYEXIT or FULL, these commands will not collect any program call/return data.

You can determine a service program’s ENBPFRCOL setting using the procedure shown in “Finding a program module’s ENBPFRCOL setting” on page 197. Using this information, the user can change the program before running a collection to ensure the appropriate program collection enablement attribute setting is in effect rather than wasting effort collecting useless data. The collected Stats data has no indication that an unenabled program ran.

- **MI Complex Instruction:** The name of the instruction that this entry is for. Library and program name columns are blank. Flyover text appears for any MI Complex you hover the cursor over. See “MI Complex Instructions” on page 240 for information about the more interesting and popular (from a performance point of view) instructions.

**Note:** Most of the programs have flyover text, but some do not. The flyover data is in the iDoctor library's QIDRGUI file QAPGMDESCS. You may add your own program names and descriptions to this file. Doing that can improve the Stats data analyst's productivity by having the various applications and vendor package's program description information readily available.

If you add your own entries to the file, make sure you save it before installing an iDoctor update and re-enter your changes. Otherwise, your input will be lost.

- **Module name:** The module name within the called program.
- **Procedure short name:** The called procedure name within the module.
The following fields are included in the report:

- **Number of threads**: The number of different job threads that called the program or invoked the MI instruction. This includes both primary and secondary threads.

You can view the job names and thread identifiers with the other default Stats Flat analysis report by Job and Program in Figure 3-12 on page 74.

- **Times called**: The number of times this row's program or MI instruction was called.

- **Calls made**: The number of times this row's program called another program. Most MI instructions always have zero in this column; there are some exceptions, but they do not affect Stats analysis.

- **MI complex instruction count**: For program entries, the number of MI complex instructions it called. It is zero for MI instructions.

- **Inline CPU usecs**: The amount of CPU time, in microseconds, used within the program or MI complex instruction.

  **Note**: The program's CPU time values are not the same as they would be if Stats collection was not running (they are higher). There is Stats collection processing cost, some of which is subtracted out of the program's CPU usage time; an exact collection process factor is not available.

  Do not compare the Stats CPU usage values against CPU times reported by other iDoctor tools or i5/OS commands. They often will not match, but the reasons are due to collection methodology and not because of the programs.

  On the other hand, you may find that relative amounts of CPU and elapsed time change (for example, before and after) are applicable to the non-data collection numbers.

  The Inline CPU usecs column is this report's default sort key in descending sequence.

  Because of its default sort key, the default view's first row represents, for the jobs/threads enabled in the collection, the program or instruction that used the most inline CPU time during the collection.

- **Cumulative CPU usecs**: The total amount of CPU time used by a program plus the CPU time used by all programs and MI complex instructions that the program called or indirectly caused to be called at lower call levels. See the Call Level list definition entry on page 107 for more information.

  For additional information about cumulative recording, see 5.7.1, “Service and non-Service program data recording” on page 124.

  **Note**: For more information about Inline and Cumulative, see 1.7.4, “What are inline data and cumulative data” on page 10.

- **Inline elapsed usecs**: The amount of elapsed run time (for example, wall clock time) used by the program or MI complex instruction in microseconds.

- **Cumulative elapsed usecs**: The amount of elapsed run time (for example, wall clock time), in microseconds, used by the program and all programs and MI Complex instructions that the program called or implicitly caused to be called at lower call levels.

  Cumulative elapsed usecs may contain, but is not necessarily limited to the wall clock time used for disk I/O waits, CPU queuing, CPU use, LPAR processor contention, socket waits, object Lock and Seize wait and other wait conditions inherent in the run time environment.
Investigate excessive or unaccountable cumulative times in more detail, first with Job Watcher. That should provide definitive results. If it does not, other iDoctor tools, such as an PEX Analyzer Trace of disk I/O or task switch are possibly needed.

Other fields
Fields not shown in the view in Figure 3-1 on page 58 are counts of the described event (for example, DB reads, software counters, and so on). Some possible reasons for high values are provided. All disk I/O counts are for physical disk operations. Logical disk I/O counts, which are counts of data base record (row) movement between main storage buffers, are not part of PEX Stats data.

There are eight types of disk I/O, data base or non-data base object, read or write, synchronous or asynchronous.

- Inline synchronous DB reads: Occur often when changing file records or randomly reading a file.
- Inline synchronous non-DB reads: Reading data areas, data queues, user profiles, journal records, loading programs, and user spaces.
- Inline synchronous DB writes: Changing a file that is open for I/O (both input and output) using the same file definition in the program.
- Inline synchronous non-DB writes: Spool file, joblogs, data areas and data queues, and so on.
- Inline asynchronous DB reads: Data base file blocked sequential reads and SMAPP physical file sampling reads.
- Inline asynchronous non-DB reads: Same as inline synchronous non-DB reads.
- Inline asynchronous DB writes: Data base file blocked sequential writes to add records to the file.
- Inline asynchronous non-DB writes: Same as inline asynchronous non-DB writes.
- Inline software counter 1, 2, 3, and 4: PEX Event counters.

**Note:** There is more information about PEX Event counters in “Collection Properties: Event Counters Definition” on page 189.

- Inline I/O pending waits: Usually the result of a wait for a previously scheduled read operation.
  
  May often be associated with System Managed Access Path Protection (SMAPP) Retuning or journal operations.
- Inline synchronous I/O waits: Usually the result of a wait for a previously scheduled asynchronous wait operation.
- Cumulative software counter 1, 2, 3, and 4: Cumulative count value of PEX Event counter 1, 2, 3, or 4

  The I/O types discussed above also apply to the cumulative values:
  - Cumulative synchronous DB reads
  - Cumulative synchronous non-DB reads
  - Cumulative synchronous DB writes
  - Cumulative synchronous non-DB writes
3.3 PEX Stats Flat view by Pgm/MI Instr: analysis

This section discusses some of the analysis you can do with the Stats Flat data's default view and with other views by resorting the data. This discussion's data view is in Figure 3-1 on page 58. This is the highest level summary view of the data. Depending on the sort key (column and descending sequence), the first few report rows represent the programs and MI instructions that had the highest (or lowest) values for the column of interest.

The default report's view sort key is Inline CPU time in descending sequence. This ordering immediately identifies the programs or MI instructions that used the most Inline CPU time during the collection period for jobs that underwent collection.

There are a number of ways to analyze the data. The number of threads column shows how many different job threads called the program. You might be interested in:

- High resource usage programs
- Knowing what programs were called the most in the application or operating system
- The use of specific MI instructions

One way to work with the data is to sort the initial view and, using one view at a time, manually scan the data. The example shown in Figure 3-2 on page 64 shows how to reorder the initial view. The result shows which programs or MI instructions were called the most (the Times Called column). This approach may not be the most efficient, and there are more productive procedures, such as modifying the SQL and letting iDoctor's server jobs do the work. More information about building your own view is in 3.4.1, “DB file Full Open analysis and iDoctor view modification” on page 75.

The collection that you are working with may or may not cover all jobs on the system; that is up to you to determine. Even if you specified to collect over all jobs, some of them may not have done anything during the collection. To discover what jobs are in the collection and whether they did much work, go to the collection properties. Use the discussion in “Collection properties” on page 187 to guide you to the collection job’s description and to see their resource usage summary.
3.3.1 Inline CPU use: the default Stats Flat analysis data view

The initial view of the data is in descending sequence by the Inline CPU time value. If the row represents a MI instruction, there is usually not much you can do to reduce its CPU use other than to use the instruction less. Whether or not that is possible depends on the application. Many times the high CPU usage MI instructions include:

- **SETCR** (Set Cursor) if the workload has high data base activity. Normally there is not much you can do about this instructions resource use.
- **RSLVSP** (Resolve System Pointer), which is used extensively throughout many system functions. Sometime very high “times called” counts affect this instructions “times called”.
- **CVTD** (Convert date) used in many applications.
- **CRTDSINX** (Create Data Space Index) used to build an index over a data base physical file. May be preceded with a number of ESTDSIKR instructions. Long running CRTDSINX instructions very often cause high job wait time. It is best to analyze this possibility with Job Watcher.
- **CRTS** (Create Space) to create temporary workspaces.
- **CRTDOBJ** (Create Duplicate Object) to open a data base file.

Some of the high Inline CPU operating system programs include:

- **QDBGETKY**, Data base unblocked read by key.
- **QDBFEOD**, Data base Force End of Data used to force a changed or new record out of the job's ODP (Open Data Path) buffer and into the data base file.
- **QMHSNDPM**, Message Handler Send Program Message. Commonly used in exception or error situations.

If a program has very high Inline CPU use, it may be worthwhile to investigate that particular program in more detail. Use the PEX Analyzer's PEX PROFILE collection and analysis to see the CPU time used within the program. PEX Profile shows the “relative” amount of CPU use by each of the program’s HLL language instructions. It does not tell you how much time an instruction takes (it is not an instruction timing function).

Quite often, there are program functions (such as RPG LOKUP or COBOL SCAN, array processing, internal sorts, unaligned or overlapping moves, short receiver variables, or loops of one type or another) that use higher than “average” amounts of CPU time. Note that PEX Profile shows the amount of CPU used by each instruction; it does not show how many times the instruction was executed.

If a program has low Inline CPU and low Cumulative CPU, it is possible that some processing can be eliminated by including the code into the calling program, perhaps by making it a program subroutine or procedure. This should reduce a high number of external calls (Call / Return). This may be the right thing to do if the program’s call count is fairly high, which implies high use of the machine’s program call/return functions.

Changing the PEX Stats initial view: resorting

The Stats Flat default view's sort key is Inline CPU in descending sequence. Any iDoctor SQL driven data view can be reordered by left or right clicking the column header. For example, in a Stats Flat view, right-clicking the Times Called column header reorders the view to descending sequence using that as the sort key. Data views can have multiple sort keys; iDoctor lists, such as the Analysis Reports or the Report Descriptions, have only one sort key.
In all cases, clicking any column header orders the view by that column's data in ascending sequence; right-clicking the column header orders the view by that column's data in descending sequence.

Left or right-clicking a data view's column header replaces all of its sort keys with a new, single key.

A data view can have multiple sort keys. To add a key to the sort sequence, hold down the shift key while you either right- or left-click the column you want added as a new sort key. This sequence can be repeated to assign multiple sort keys.

**Note:** A sort key has two properties, the field (or column) to sort and the sort order (ascending or descending).

The discussion from here to 3.3.4, “Cumulative Elapsed Time or CPU usec” on page 70 focuses on minor modifications to the default view.

Following that, there are more complex analysis views that require changes to select, add, sort, and group by some of the data fields.

In this example, the result in Figure 3-2 shows that the highest Times Called value was for the MI complex instruction *MATINVIF.

**Performance Note on building up sort keys:** If you have a large view (thousands of rows) and you want to sort it by multiple keys, the best performance is achieved by using iDoctor's **Query Definition menu → Sort** tab and defining all the sort keys before doing the sort. See “Specifying sort keys” on page 210 for more information.

### 3.3.2 Times called

There are other data views that can help the analysis process. One of these is to change the view from its default Inline CPU order to some other key(s).

The discussion from here to 3.3.4, “Cumulative Elapsed Time or CPU usec” on page 70 focuses on minor modifications to the default view.

Following that, there are more complex analysis views that require changes to select, add, sort, and group by some of the data fields.

In this example, the result in Figure 3-2 shows that the highest Times Called value was for the MI complex instruction *MATINVIF.

![Figure 3-2 Sort by Times Called column in descending sequence](image-url)
Chapter 3. Analyzing PEX Stats Flat data

Times Called shows:

1. Whether or not a particular program or instruction was used and, if so, how many times.
2. If the Times Called value is very high and both the calling and called programs are application programs, consider embedding the called program into the caller.

While this may result in additional maintenance cost, it may also result in a significant savings in call/return overhead; you and your customer have to evaluate the trade-offs.

Using multiple sort keys

Further investigation shows (in Figure 3-3) that there are rows with the same Times called. For this example, let us explore the rows where the Times Called value is 139. Using 139 is arbitrary; the point is that there may be a row with a number of equal data values and you need to do more investigation using one or more additional columns.

![Data Viewer - 1 - [RTURNER/StatsFlat/Summarized CPU and IO by Pgm/Inst - #1]](image)

Figure 3-3 Stats Flat view ordered by Times Called descending

There may be questions about the significance of the same call count in all those lines. For example, for all jobs with the selected Times Called count, which of them used the most Inline CPU time?

Add Inline CPU use as a second sort key and refresh the view. To set the sort sequence to Times Called and Inline CPU time:

1. Right-click Times called. This sorts the view by Times Called in descending sequence.
2. Hold down the Shift key and right-click Inline CPU usecs: The shift “adds in” a descending (right-click) second sort key (Cumulative CPU usecs).
The result looks like Figure 3-4. This shows you immediately which procedure in the group with equal Times Called had the highest Inline CPU use (procedure NSFDbClose).

This is a somewhat trivial example because you could see the result by viewing one window of data; when you have larger quantities of data, it is an easy way to reduce them to a manageable number of entries.

There is more discussion on this topic in 3.4.2, “Program behavior analysis using Stats Flat Times Called values” on page 79.

---

**Figure 3-4   View sorted by two columns**

**Query Properties: what is the views Sort Sequence**

When working with iDoctor data views, you may occasionally need to know what a view's sort sequence is.

iDoctor uses SQL to generate the view and you can view the SQL statement's Order By parameter using the view's “query properties”. Do that by:

1. Right-click anywhere in the data view.
2. Select **Properties** from the drop-down menu.
3. Select **Query** from the sub menu.
Chapter 3. Analyzing PEX Stats Flat data

Getting to the query properties

Figure 3-5 Getting to the Query Properties view

Selecting Properties → Query provides the view in Figure 3-6. The top part contains the library name, the collection name, and the view's Report Description. The middle part contains the SQL statement that generates the view. The bottom part shows the data's Library, File, and Member name.

In this case, G_STATSFPG is the name of the PEX Stats Flat analysis output file in the collection library. Note that the view's member name is different than the collection name. The view's member name is generated by the analysis program. If you write your own SQL or some other method of processing the data, use the analysis view's Properties to determine the member name.

Figure 3-6 A View's Query Properties (sort fields and sequence)

The “ORDER BY” values in the SQL statement SELECT * FROM libname/G_STATSFPG ORDER BY QSTINV01 DESC, CUMCPU01 DESC show the sort (order by) field(s) and sequence(s) to be:

1. QSTINV01 (times called) DESC (descending)
2. CUMCPU01 (Cumulative CPU usecs) DESC (descending)
View the data field's names and descriptions by selecting **Query Definition → Field Selection**, as shown in Figure 3-7:

1. Right-click in the view.
2. Select **Query Definition** from the menu.
3. Select **Field Selection** from the submenu.

**Query Field Selection**

![Data Viewer - #1 - [RTDHDATA/M1512a/Summarizr...](image)](image)

Figure 3-7  Getting to the Query Definition's Field Selection menu
This sequence presents the view in Figure 3-8. Its columns are:

- **Show**: A check box that allows you to include or omit a field from a view.
- **Field Description**: What the field represents.
- **Field**: The name of the field.
- **SQL Expression**: If the field is generated from other fields, the SQL expression that was used.

Note that the Field Description and name are the same as the data provided by the DSPFFD (Display File Field Definitions) command over the file. For a list of the Stats Analysis output files, see “PEX Stats data analysis output files field names: V5R3 / V5R4” on page 244.

![Figure 3-8 Query Definition Field Selection menu](image)

### 3.3.3 Inline Elapsed Time

The Inline Elapsed Time value represents the wall clock time spent in the program or MI instruction represented by the current row.

If the row's ratio of Inline Elapsed to Inline CPU is high, the program may be doing a lot of disk I/O. You can modify the query to calculate and sort on this ratio.

Another possible cause of a high Elapsed/CPU ratio could be that the program encountered a lot of waits on objects or gates. More information can be gathered about these types of wait times using Job Watcher.

If a program's inline times are very low, it usually is not making a significant contribution to the Cumulative times. These program's can be omitted from the views by selecting only those whose Inline... times exceed some amount. As a starting point, omit those whose elapsed time is less than ten percent of the application program's cumulative elapsed time. You may have to come back to them later but initially it's more productive to leave them out of the view.
3.3.4 Cumulative Elapsed Time or CPU usec

High Cumulative Elapsed Time and Low Inline Elapsed Time values may indicate that the row you are viewing probably is not the one in which you are interested.

High Cumulative values indicate that more Stats data and analysis is needed using Stats Hierarchical collection over the program's jobs. You can find the program to job relationships with the report discussion starting at 3.4, “PEX Stats Flat Job level default report view” on page 74.

Note: MI Instructions do not call anything else, so their Cumulative and Inline values are almost always equal. There are some situations in which this is not always true, but they do not affect a Stats analysis.

3.3.5 Markers: program name analysis

Sometimes it is necessary to view the call frequency and resource cost of specific MI programs and MI Complex instructions. Normally, the approach is to use iDoctor's Query Definition Record Selection to modify the SQL and the result view.

At this point, the Stats data analysis discussion moves to a somewhat more technical level because the view has to be modified.

The following introduces you to changing the SQL. For more complete detailed information about modifying the view and building your own user defined SQL views, see “iDoctor’s Query definition interface” on page 202.

For example, assume there is a need to change the default Stats Flat by Program view to find usage information about only the operating system’s data base file open and close programs. More specifically, the objective is to determine how many times the full file open and close operations occurred. There are two ways to approach this.

1. Sort the data in program name sequence (select the **Program Name** column header) and scroll through the data looking for what you want.

2. Right-click the view and select **Query Definition → Record Selection** tab to change the SQL so the view contains only the information about the programs of interest. This is the approach to take in a general case. When the changes are complete, you can save the SQL as a User Defined Query and reuse it later rather than building it anew every time it is needed.

Once you are familiar with the Stats data, this method will be much faster and easier to use rather than manually sorting and scanning the data. For more information about building your own SQL, see the following sections:

- “Accessing the Query Definition interface” on page 202
- “Selecting specific input data” on page 209
- “Saving a new query definition” on page 215

The following is an example of changing the SQL. The sequence of operations is:

1. Bring up a view of the data that contains what you are looking for.

2. Right-click in the view and select **Query Definition → Record Selection**. This brings up the Query Definition menu with its Record Selection tab enabled.

   Use the following information to enter the program name selection criteria. The results are shown in Figure 3-9 on page 71.
3. Open the **Field** list and select **PGMNAM**.
4. Open the **Operator** list and select **List**.
5. Place the cursor in the Value list and enter the text ‘QDBOPEN’ ‘QDBCLOSE’.

**Note:** Note the syntax for one or more string values: single quotes around each value and a single space between the values.

6. **Query Definition Record Selection** requires this syntax to build a SQL Where clause.

   This example assumes you know what program names or other search arguments to use. This is not always the case, and you may have to take a different approach. For example, you may want to select other columns whose data values meet specific criteria, such as the Inline CPU being greater than 10 seconds or the Cumulative Elapsed time being greater than 120 seconds, and so on. From this view, select the programs of interest and use them as selection criteria in the **Query Definition → Record Selection** specification.

7. Click **Add item** and the SQL parameters appear in the Record Selection Filter List. You can iterate through steps 3-6 to apply multiple sets of record selection specification.

8. Click **OK** to exit and run the query.

The above is a short specific example. For a more in-depth discussion about building and using your own queries, see “Querying PEX Stats data using iDoctor’s GUI” on page 202”.

![Figure 3-9 Query Definition final menu](image)

**Markers: i5/OS programs**

Most of the operating system programs do not exhibit poor performance or high resource usage. However, normal use of them can be a “marker” or indication of directly or indirectly related activity in other programs that does impact performance.

Some of these programs are discussed in “i5/OS programs” on page 236.

If these programs appear in your data, more analysis is needed to determine if they had an effect on performance.
3.3.6 Markers: MI Complex Instructions analysis

In normal processing, most high MI Complex Instruction Times-called counts occur in fast instructions such as RSLVSP and are not a problem. On the other hand, there are some MI Complex instructions that are “marker” instructions. These are not necessarily high cost in and of themselves, but when they are used by some functions, their presence can indicate possible performance impact.

Some of these programs are discussed in “MI Complex Instructions” on page 240. If some of these instructions appear more than a few times in your data, it is worthwhile to do more analysis to determine their effect on performance.

You can also analyze the use of various classes of instructions. A class in this context means all instructions with the same first three characters of the name. This applies even if their Times Called count values are not very high. This includes Creates (CRT), Destroys (DES), Activates (ACT), Deactivates (DEA), and Estimate DSI Key Range (EST). If there is Create/Delete occurring on Programs or other objects, such as Cursors, Programs, Files, Activation Groups, Dump Spaces, and so on, investigate them. See the sample view in 3.3.7, “MI Create xxx Instruction Analysis sample view” on page 72.

High call counts often occur in data base MI Instruction access functions, such as RETSDSE, UPDSEN, and so on, and usually are not a problem. However, the mix of instruction usage may indicate that study is needed to determine how the application processes its data base records and how data base file maintenance is done.

Investigate physical files that have an excessive number of deleted records, a number of seldom used logical files, or for which the application’s most used processing order is not the same key as the file’s organization.

3.3.7 MI Create xxx Instruction Analysis sample view

This section discusses and demonstrates collecting information about classes of MI Complex instructions.
The data in Figure 3-10 is a consolidation of the collection’s MI Create instruction’s use. To
determine its usage rate, get the collection run time from the collection’s properties and
calculate the items call rate.

Figure 3-10 MI *CRT... instruction’s usage example

Figure 3-11 shows the SQL for this view. To look at other instructions or classes of
instructions, change the WHERE MICPXNM LIKE ‘*CRT%’ clause argument to a different
value.

Change LIKE ‘*CRT’ to LIKE ‘*DES’ and rerun the query to see all of the MI Destroy xxx
instruction usage.

```
SELECT MICPXNM, sum(qstinv01) AS TIMESCALLED, sum(inlelp01) AS INLELP,
    sum(inlcpu01) AS TOTCPU,
    sum(qiisdr01+qiisnr01+qiisdw01+qiisnw01+qiiadr01+qiianr01+qiiadw01+qiianw01) AS
    TOTDIO, sum(qstinv05) AS NBRTHREAD, QCPLNM FROM libname/G_STATSFPG WHERE
    MICPXNM LIKE '*CRT%' GROUP BY MICPXNM, QCPLNM ORDER BY INLELP DESC
```

Figure 3-11 SQL for MI *CRT... instruction usage

3.3.8 Considerations for Selection, Start/End timing, and service programs

Sometimes the data does not show what you expected, which may depend on the run time
environment. For example:

**Selection**  
Is it necessary that all jobs be enabled for collection? If you know
which jobs are causing the problems, you can limit the selection
criteria to collect only certain jobs data and ignore other jobs.

If there is a high number of active jobs, collecting PEX Stats data can
cause significant performance degradation. If it is possible, consider
subsetting the collection over a smaller but representative set of jobs.

**Start and End timing**  
To ensure that you get complete data for all programs in a collection,
start PEX collection before the test programs start and end PEX
collection after the test job is complete. Following this procedure
ensures that you will capture all test program activity.

**Missing Program**  
If you know that service programs are in the job mix but did not appear,
either they were not used or, more likely, they are not properly enabled for collection.
In the latter case, the program is Unhooked and the collection functions did not recognize the program Call. This occurs most frequently with ILE Service Programs because the service program's developers did not change the program creation command's default from ENBPFRCOL(*PEP) to ENBPFRCOL(*ENTRYEXIT).

To correct this, change (CHGSRVPGM) or recreate (CRTxxxMOD) the service program(s) using ENBPFRCOL(*ENTRYEXIT) so that PEX Data Collection can intercept its Calls and Returns.

This can occur with any ILE programs (not just service programs) that make extensive use of modules and procedures (C and C++ programs are prime examples).

### 3.4 PEX Stats Flat Job level default report view

Figure 3-12 shows the PEX Stats FLAT report by Job/Program/MI Instruction default view.

![PEX Stats FLAT report by Job/Program/MI Instruction default view](image)

The Stats Flat job level report contains the same data as the PEX Stats FLAT report by Program/MI Inst report plus additional columns that describe:

**Job name, user, number**

The job’s fully qualified job name.

**Job Thread id**

The job’s full thread id.

**Initial Thread Y or N**

Indicates if this entry represents the jobs primary thread. (if Y = yes, then it is the primary.)

G_STATSFJB is the file name of the Stats Flat analysis output file. This file’s field names and descriptions are shown in “Stats Flat Job file G_STATSFJB” on page 245.

Because the file contains both job and program information, the analysis can direct you to the jobs that need additional data collected, such as a Stats Hierarchical collection. In some cases, the Stats Flat Job/Pgm data can be used to determine specific degraded performance activity and relate it to specific jobs and programs.
3.4.1 DB file Full Open analysis and iDoctor view modification

The following section continues the discussion from “Markers: i5/OS programs” on page 71 to focus on specific i5/OS program usage.

A high rate of data base file Full Open and Close operations usually impacts system performance. This can often be corrected at a minimal cost by reducing the number of times the application programs call the system’s data base Full Open and Close programs. Finding whether or not the Full Open/Close counts are too high is a two step process:

1. Use PEX Stats Flat to determine whether or not there are high file Open/Close rates. From the Stats Flat Analysis results, change the default analysis’ program data view to select only the QSYS library program’s QDBOPEN, QDBCLOSE, QDMCOPEN, and QDMCLOSE.

2. Determine if the Times Called count value is high for QDBOPEN or QDBCLOSE. (In this case, consider more than 50 per second as too high.) Another, alternative objective is keep it low enough that no user ever experiences spikes in performance. If it is too high, then it is necessary to determine what job(s) they happened in and what application program is calling them.

Note: This discussion is about the application's steady state case after all jobs are started and have finished initialization, including opening their data base files. This investigation can uncover the poorly implemented programs that were written without sufficient concern for optimum performance during peak load processing.

You can find which jobs are doing the most opens by looking at the Stats Flat job level analysis report view for the jobs that are invoking the i5/OS data base file Full Open/Close programs. See Figure 3-17 on page 78 for an example output from this type of analysis.

There may be explicit indications in the Stats Flat program level data about what application program is doing a high number of data base file Full Opens and Closes. This is discussed in more detail in 3.4.2, “Program behavior analysis using Stats Flat Times Called values” on page 79.

In any case, the calling program can be found using Stats Hierarchical data collection and analysis.

Viewing specific data: modifying the View Query’s Record selection
A view’s SQL can be modified using the iDoctor’s Query Definition menus. Start by:

1. Bring up a view of the PEX Stats Report Description “Jobs resource usage summary view ordered by total inline CPU usage” (Figure 3-20 on page 82).

2. Right-click in the view to get the Query Definition menu.
3. Select the **Record Selection** entry (see Figure 3-13).

![Figure 3-13 Accessing the Stats data change record selection function](image)

Figure 3-13 Accessing the Stats data change record selection function

In the Record Selection menu (Figure 3-14), do the following on the rows indicated by the arrows:

1. Open and select the **Field** list entry to apply the selection criteria to. All of the Stats data fields are in the list. In this case, select **PGMNAM**.

2. Open the Operator list and select which one to use. There are a number of options. In this case, use the LIST operator to select one out of four possible values.

3. Fill in the selection values. In this example, there are four program names of interest. Enter them as text strings;
   a. Each value is a separate entry.
   b. Each entry is enclosed in *single* quotes.
c. Entries are separated from each other by a single blank. The entries are entered as shown in Figure 3-15.

```
'QDBOPEN' 'QDBCLOSE' 'QDMCOPEN' 'QDMCLOSE'
```

*Figure 3-15  *iDoctor Query Definition Record Selection text string entry format

After entering the selection criteria text string, proceed as follows (and as shown in “iDoctor Query “Record Selection List” Entry capabilities” on page 77).

4. Click **Add Filter** to add the selection criteria to the query.

5. The result of Add Filter is that a new line appears in the Record Selection Filter list. Internally, iDoctor converts this into a SQL Select statement’s Where clause when the query definition is complete and **OK** is selected.

The Record Selection Filter List can have multiple selection rows related to each other with AND and OR specifications.

You can group related rows by marking contiguous rows and selecting the **Parenthesis** button. You can remove parenthesized groupings by selecting them and selecting the **Parenthesis** button, which clears the parentheses.

Selecting a Record Selection Filter List row enables the Update and Remove buttons. There is an option to Remove All the entries or you can select and remove all selected rows.

You can move a Record Selection Filter List row to the end of the list using the following procedure:

1. Select the row.

2. Select **Add Item**. This adds the row to the end of the list.

3. If you want to change the original row, go back and select it (the row you highlighted in step 1).

4. To delete it, select the **Remove** button; to change it, modify its parameters and select **Update**.

This procedure provides a way to insert a selection list entry and move the existing entry to the end. There are no iDoctor row move up/down functions for Record Selection Filter List entries.

**iDoctor Query “Record Selection List” Entry capabilities**

iDoctor converts the list entries to SQL Select Statement Where clauses when **OK** is selected. There are a number of Record Selection capabilities. You can:

- Use multiple selection rows.
- Use AND and OR rows together.
- Nest the AND’d and OR’d rows.
- Enclose continguously marked rows in parenthesis.
- Nest the parenthesized rows.
- Select and update an individual record select row.
- Remove all or all selected rows.
Figure 3-16 shows the Record Selection program selection steps 4 and 5.

![Query Definition](image)

**Figure 3-16** Query Definition > Record Selection: program selection steps 4 and 5

Clicking **OK** runs the SQL query. The result view contains only the rows that meet the selection criteria.

For this example, Figure 3-17 hides the extraneous fields so it fits in this document. The visible fields apply to data base file Full Open/Close processing and analysis.

![Data Viewer](image)

**Figure 3-17** Modified query program selection results: full file Open/Close job candidates

After analyzing the first set of data and gathering the job ID values necessary for more analysis, the next step is to run a Stats Hierarchical data collection with the identified jobs enabled for collection. If file open and close occurs in these jobs, the collection's data identifies which job they occurred in, which user programs called the i5/OS Open/Close programs, and how many times each was called.

See Chapter 4, “Collecting PEX Stats Hierarchical data” on page 89 for information about collecting and analyzing PEX Stats Hierarchical data.
3.4.2 Program behavior analysis using Stats Flat Times Called values

There are MI instructions that, if they are called often (for example, they have a large Times Called value), can be an indication of degraded performance as a result of improper program implementation. One of the most frequently occurring situations is the program that is called many times within a job and each time it is called it goes through full initiation and termination.

What is full initiation? The first time an application program is called, it allocates its work areas and opens its files. Proper implementation ensures that if the program returns and may be called again that it returns to the calling program but leaves its work areas intact and its files open in anticipation of the next request. When it does this, it is considered to still be activated, because its work areas still exist after it returns. There is a significant performance advantage for a program to work this way; subsequent calls are very fast. Subsequent calls are not fast if it closes its files and deallocates its work areas every time it returns to its caller.

This type of behavior is sometimes evident in Stats Flat data; sometimes it is necessary to use Stats Hierarchical data to capture all of a job’s program initiation and termination activity.

The Stats Flat data in Figure 3-18 on page 80 shows two jobs that are very likely candidates for this type of performance improvement. The SQL for this view was built after identifying higher than expected Times Called counts of some application programs and the i5/OS programs QRGXINIT, QRNXINIT, QDBOPEN, and QDBCLOSE (OPM and ILE RPG program initialization, data base Full Open and Close) and the MI instruction *DEACTPG (Deactivate Program) in some jobs.

What you are looking for, by job, are the application programs whose Times Called count matches or is very close to the Times Called counts for program QRGXINIT (for OPM programs) or QRNXINIT (for ILE programs) and the MI instruction *DEACTPG.

You may also see that data base open and close counts are very close to an integer multiplier of QRGXINIT and *DEACTPG times called counts. That integer may be very close to the number of files that the program has declared and opens each time it is called.

The SQL in Figure 3-19 on page 81 shows information selected from two jobs that had high program initialization counts. These were jobs with job numbers 833244 and 834059. The views are edited to ensure data confidentiality.
The results show, in both jobs, the program P07 Times called count values were the same (or very close) as they were for initialization (program QRGXINIT) and termination (MI Instruction *DEACTPG). This correlation puts a very high probability that program P07 needs changes to its return methodology to improve its performance.

Note also that in the 833 job the P07 program’s data base open and close counts are twice the program’s call counts and in the 834 job the ratio is also about two to one. It is valid to interpret this as meaning that the program is very likely to be doing a Full Open of two data base files every time it is called.

These correlations almost ensure that program P07 is a likely candidate for changes that would eliminate all but the first call’s initialization processing. Usually the fix for an RPG program is to change it to use the RPG RETRN operation with LR OFF (Last Record indicator). This technique has been discussed in performance related literature, classes, and presentations for the i5 and its predecessor systems for a long time, but the problem still persists. Newer language implementations are not immune either.

Because there can be a lot of files opening and closing as well as work space creation and deletion, this program, which has high call counts in multiple jobs, is often a large contributor to unnecessarily high resource usage and directly and indirectly affects system performance.

Part of the direct effect is its higher than necessary CPU time and number of disk operations. Its indirect effect is high use of Auxiliary Storage Management (ASM) functions, which can cause delays due to seizures and SLIC waits. Indirect costs are best identified and evaluated with Job Watcher or possibly with PEX Task Switch Trace analysis.
Chapter 3. Analyzing PEX Stats Flat data

Other open/close effect analysis
The difference between a row’s Inline and Elapsed run time values can be a key factor in analysis.

One cause could be disk I/O wait time. Stats data shows the number of disk operations issued by the program (the inline value) and down the call stack (the cumulative value). It does not provide the disk I/O time.

Another possibility is object, event, or other types of waits.

If you suspect any of these are happening, you may be able to get more information by including Event counting in a Stats collection (see “PEX Event Counts Collection specification” on page 178). If that does not provide definitive information, the best “next step” is to run Job Watcher and focus on the job wait information.

Another analysis that should be done is to look at the ratio of the times called counts of the program QDMCOPEN to the programs QDBOPEN and QDBOPENC (the command processing program for a CL program OPNDBF operation). QDMCOPEN is a common data base routing program that is the gateway to all open processing.

The ratio is going to be one or higher. If it is equal to one, then all calls to QDMCOPEN are for a full data base open. If the value is greater than one, then there are other types of open requests. Some of the other open types are spool files, communication files, “shared open” data base files, and so on. The primary performance impacts are associated with the high use of data base Full Opens and that is why there is so much attention given to QDBOPEN. There could be other high cost opens happening, and this is one way to quickly discover them.

Look at data base file Full Open AND Close frequency? For the whole system, more than 50 to 100 per second peak is too many.

Use Collection Properties to determine the run time and calculate the Open and Close rate.

Look at Figure B-7 on page 188. Use the sum of a job’s Full Open and Close counts divided by its elapsed time to calculate its open-close rate per Second.

Select QTSJNB, LIBNAM, PGMNAM, MICPXNM, QSTINV FROM libname/G_STATSFJB WHERE ((qtsjnb like '833244%') or (qtsjnb like '834059%')) and ((LIBNAM LIKE 'Q%' AND PGMNAM LIKE 'QRGXINIT%') OR (LIBNAM LIKE 'Q%' AND PGMNAM LIKE 'QDBOPEN%') OR (LIBNAM LIKE 'Q%' AND PGMNAM LIKE 'QDBCLOSE%') OR (MICPXNM LIKE '*DEACT%') OR (LIBNAM NOT LIKE 'Q%' AND MICPXNM LIKE ' %')) and QSTINV > 100 ORDER BY QTSJNB ASC, QSTINV DESC

Figure 3-19  SQL for analyzing times called values view

When you use this SQL template, change the libname (FROM statement’s file library) and qtsjnb (job number) values.
One way to “fix” the high rate of data base full file open is to use what are called Shared Opens, which have very little Open/Close processing cost. They use the i5/OS program QDBSOPEN. The customer may be aware of this approach and has implemented it. If so, you may see a number of calls to program QDBSOPEN.

For more information on Shared Open, go to:


3.4.3 Job resource usage summary view

Figure 3-20 shows the job resource usage summary view.

![Figure 3-20](image)

This very high level view summarizes every job’s resource usage and other values from a Stats Flat collection.

To drill down further into the high cost jobs, look at the jobs that are using the most CPU time or doing the most disk I/O or other values and discover what functions the programs perform. There could be some surprises there when they are identified. Investigate in more detail the jobs that fit this type of usage.

1. This is a summary view. One follow-on approach is to query the same file that this view is over and look at the job’s individual program’s data values.

2. If the previous step is not satisfactory or does not lead to a solution, do Stats Hierarchical collection and analyze specific jobs to identify the program call/return flow and resource usage in the job’s programs.

This view shows that there are 349 jobs in the collection (as shown by the count in the view’s lower right hand corner). There are usually very few jobs that use a significant amount of resources. Look at descending sort views of both the Total Inline CPU usage and disk activity (Total DIO) to find the most active jobs.
3. On systems in which there is a lot of SBMJOB activity and a very high number of the submitted jobs have a very low run time (the job Total Inline ELP column), one way to improve overall performance is to reduce the number of jobs that are starting and stopping. One way to do this is to change the jobs to be data queue message driven. Short lifetime jobs can be seen by modifying the SQL to show only the jobs whose highest cumulative elapsed time is less than a few seconds (generate, limit and sort by the SQL MAX function on cumulative elapsed time and Group by job name/user ID/number/thread).

4. Compare the jobs CPU and Elapsed time to see if there is very low CPU use and high elapsed time.
   a. This can occur in jobs that are event or message driven and spend a lot of their time waiting for new work.
   b. It can also happen in jobs that have a lot of synchronous disk I/O wait time, I/O pending waits on asynchronous reads, or waits for “other job” initiated page fault I/O (pending faults).

   One of the best ways to analyze these particular types of wait times is with Job Watcher.

   **Example 3-1**  SQL for the Jobs resource usage summary view in Figure 3-20 on page 82

   ```sql
   SELECT QTSJNM, QTSJUS, QTSJNB, QTSTIF, QTSTHI, MAX(CUMELP) AS JOBELPTIME, sum(inlcpu) AS TOTINLCPU, sum(inlelp) AS TOTINLELP, sum(qiisdr+qiisnr+qiisdw+qiisnw+qiiadr+qiianr+qiiadw+qiianw) AS TOTDIO FROM libname/G_STATSFJB GROUP BY QTSJNM, QTSJUS, QTSJNB, QTSTHI, QTSTIF ORDER BY TOTINLCPU DESC
   ```

3.4.4 The ratio of Cumulative to Inline

The Cumulative and Inline values topic in 1.7.4, “What are inline data and cumulative data” on page 10 applies to this discussion.

Looking at the ratio of Cumulative to Inline values of a variable (CPU, elapsed time, and disk I/O counts) can help jump-start your analysis work by highlighting possible areas for additional analysis.

What do you do with these values when taken in an appropriate context? Consider the following:

   If the ratio of Cumulative CPU use to Inline CPU use is close to 1, what does it mean?

   If most of the CPU time was used in the program/procedure, it implies that the program probably did not call many other programs or use very many long running MI instructions.
If you are looking at the ratio of Cumulative CPU to Inline CPU usage and a program’s Inline CPU use represents a significant part of the cumulative time, additional analysis with program debug, PEX PROFILE, or PEX TRACE TPROF might be useful. The CPU time may be used within the program instructions, but that is not always the case. Neither TPROF or PROFILE always provide a definitive answer as to where all of a program’s Inline CPU time is consumed.

**Note:** PEX PROFILE is another component part of PEX Analyzer along with PEX STATS and PEX TRACE (which includes TPROF).

There are some scenarios in which the Inline CPU value is a false positive and what the view indicates is not really the case and the time is not being used within the application program. This can happen when:

- A program calls an unenabled Service Program. However, these can be unearthed, in part, with TPROF. “In part” because TPROF is a sampler, not a recorder of total values.
- A program spends a lot of time in LIC Assist routines, such as string or date and time handling.
- There is a lot of exception handling caused by the program, such as arithmetic, data, or page fault exceptions.
- If the cumulative to inline ratio is a large value it may mean that you need to look for the “downstream” programs that the program called or caused to be called by programs further down the call levels. See 5.11, “Drilling down: what used analysis” on page 130 for more information about this type of analysis.
- If the Program or MI Instruction function is to wait, then what? Note it, and if it is familiar, just keep going. If it is not familiar, discover what it is by asking the customer, programmer, and so on.
- If the cumulative to inline ratio is less than one, it should not happen unless there is a collection or analysis error.
The view in Figure 3-21 was generated by the SQL in Figure 3-22. It contains the library and program name, thread count, call counts, inline and cumulative CPU, and inline and cumulative Elapsed time values as well as the Cumulative to Inline values ratios.

If the Cum Inl ELP ratio is 1, the program does not call any other programs or use any MI Complex instructions. If it is greater than one (as shown for example in the entry for program QDBCRTME that is six lines up from the bottom), there is a lot of time being spent elsewhere in called programs or MI instructions.

If the Cum Inl CPU Ratio columns value is close to one, almost all of the CPU time was used by the program. If it is higher than one, more information is needed to help determine what programs were called. Order the view by the ratio descending. That provides a much faster way to compare the Cumulative to Inline CPU times.

The next step is usually to collect PEX Stats Hierarchical data over the job(s) that call the program that has the high ratio.

```
SELECT LIBNAM, PGMNAM, QSTINV05, QSTINV01, QSTCCT01, QSTXCT01, INLCPU01, CUMCPU01, dec(cumcpu01/inlcpu01,9,2) AS CUMINLCPU, INLELP01, CUMELP01, dec(cumelp01/inlelp01,9,2) AS CUMINLELPRATIO, QCPLNM, PGMDSC, QPRPNM FROM libname/G_STATSFPG WHERE MICPXNM = '' ORDER BY INLCPU01 DESC
```
3.5 Generating your own custom tables and graphs

For more information and a description on using iDoctor's Query Definition capabilities, see Appendix C, “Querying and graphing PEX Statistics data” on page 201.

Specific user defined view examples are shown in Appendix D, “PEX Stats user-defined queries” on page 225.

3.6 Summary of Stats Flat data analysis

Here we summarize the Stats Flat data analysis.

Stats Flat analysis can be proactive
Use it during development and testing:

- If you know what potential or high cost stuff to look for or not. It can collect the data you need to find.
- To identify jobs and program that cause high cost results.
- It is much less expensive to make changes early in a program’s life rather than later, especially after it is shipped to customers.

Stats Flat analysis can be reactive
When there are performance problems, use PEX Stats or Job Watcher. Use:

- PEX Stats if you want to see what programs are running and what they are doing.
- Job Watcher to see job level costs, such as CPU and DIO, high wait time, and object access activity, such as SQL statements, job call stack from top (i5/OS) to bottom (SLIC), and everything in between.

Stats provides a total systemwide (actually collection wide) view of program and complex MI instruction usage by job. It provides program to program flow activity in hierarchical mode.

- Job Watcher shows the jobs that wait and, for specific collection conditions, some of the object/gate holder(s).
- Job Watcher often shows the objects and events being waited on, including when the job watcher sample occurs while the job is waiting for an I/O operation to complete.
- Job Watcher does not show all called programs. If the job is in a wait at the end of a sample interval, it shows the current call stack entries at time of wait (at the expiration of the sample time). The Job Watcher call stack includes all MI programs and SLIC programs at the time of the sample.
- Service program collection enablement is not an issue with Job Watcher or PEX Trace TPROF; it is with PEX Stats, i5/OS Start Trace, and PEX Trace MI Program Events.
- Job Watcher collects and shows event counts by job and interval, while PEX Stats collects and shows totals by program at the end of collection. Program counts can be rolled up into job counts from the data by using a Group By summary SQL Select query. Figure 3-20 on page 82 is an example of this type of query.

You may find the solution or may have to collect additional data
Stats Flat data analysis can, depending on the type of problem, guide you towards a solution. Do not try to use PEX Stats to figure out disk arm utilization or page fault rates or other things that it is not designed to handle. Select the right tool to start with.
Its effectiveness depends in part on the amount of available applicable information that applies to the problems. If you know that only one program runs in a specific job and the job calls hundreds of high cost programs or function requests, you know which job the changes need to be made in. If there are multiple programs in the job, do additional PEX Stats Flat and Hierarchical data collection and analysis to determine which of the programs to change.

On the other hand, if there are high cost programs that you do not recognize in a Stats Flat collection, use the program's job identification data as selection criteria for a Stats Hierarchical collection.

**Be sure of your of date before you act.**

One set of data may show performance degradation in certain jobs and programs. You should verify it with one or two additional collection sessions to make sure you are targeting the right place before committing resources to making changes.
Collecting PEX Stats Hierarchical data
4.1 Preparing for collection

This section discusses the steps for defining, collecting, and analyzing Stats Hierarchical data.

It is assumed that the reader is familiar with iDoctor’s GUI, understands the basics of using it to interface with PEX Analyzer, and can proceed without a lot of step-by-step instruction through the process. Parts of this section may look familiar if you have studied Chapter 2, “Getting started: using Stats Flat” on page 17. This section has some differences from the Stats Flat discussion.

Stats Hierarchical collection requires the user to identify which jobs are to be selected for collection.

Because Stats Hierarchical data collection overhead can severely inhibit system performance, it is up to you to ensure that the number of concurrent jobs undergoing collection is limited. There is not a PEX Stats enforced limit to the number of concurrently active jobs undergoing collection. One of the effects of improper generic job selection is to drive the number of concurrent collections too high.

If this happens, the customer will most likely experience immediate and noticeable overall system performance degradation, especially by interactive users.

The recommendation is to keep the number of concurrent collections at twenty or less.

4.2 Create a PEX collection

The steps for creating a Stats Hierarchical collection are the same as they are for creating a Stats Flat collection. A PEX Definition must exist and is selected during the Start collection menu processing. Use iDoctor to start and stop collection, and to submit the initial data analysis.

4.2.1 Basic path

You can use either the Basic or Advanced path to create a collection. One difference between them is that the Basic path creates a PEX Definition for you, while the Advanced path lets you select an existing PEX Definition.
To start the process, right-click iDoctor PEX Analyzer in its main window and select Create PEX Collection from the sub-menu (Figure 2-14 on page 30). The menu in Figure 4-1 appears and contains a Select desired mode option box with Basic and an Advanced mode option buttons.

Select the Basic option and click Next (see Figure 4-2).

To select a PEX Statistics collection, check the Excessive CPU utilization option's button on the Problem Type Selection window shown in Figure 4-2.
Selecting **Next** takes you to the second Problem Type Selection window shown in Figure 4-3.

![PEX Collection Wizard - Problem Type Selection](image)

*Figure 4-3  Problem Type Selection: Program with Call Flow for Stats Hierarchical*

To collect PEX Stats Hierarchical data, select the **CPU statistics by program/procedure/method with call flow** button and click **Next**. That takes you to the Options window shown in Figure 4-4 on page 93.

### 4.2.2 Common menus

Figure 4-4 on page 93 shows, when you are in the Basic path, a PEX Definition named *STATSHIER* that is used to fill in the Definition box.

Skip the following three steps if you are using the basic path.

This is the first window in the Advanced path. In this window, you must:

1. Select the **Definition Type** user defined button,
2. Open the **Definition** list.
3. From this list, select a user-defined PEX Definition to be used by your data collection.

*Note:* It is not a good idea to use the name my_colln or other generic collection name.

The reason is that at some earlier time a PEX Collection may have used the same name for a collection. If it was a PEX Stats collection, its data will be lost.

It may not have been a Stats collection; it may have been a PEX Profile or PEX Trace. If you replace that collection with a different type of collection, the integrity of the previous user's data and yours to come is in jeopardy.

It is better to have your own unique (and descriptive) collection name even though it may be harder to remember.

In either the Basic or Advanced path, fill in the:

1. Collection name, if it is not already set to what you want.
2. The name of the library the collection will be put in.
3. Collection's optional descriptive text.
4. Collection time duration in minutes.
5. If you want to set a scheduled start time for the collection, click the Scheduled start time Configure button. Follow the scheduling windows instructions.

![Collection Wizard Options window](image1)

Figure 4-4  Collection Wizard Options window

After setting the parameters and options values, select **Next** to go to the Job/Task Options window (Figure 4-5).

![Collection Wizard “Job/Task Options” window](image2)

Figure 4-5  Collection Wizard “Job/Task Options” window

**Collection Job selection**

The Stats Hierarchical Job selection: in the iDoctor’s interface using the Basic path allows only Selected jobs. You must select one or more jobs to collect on.

If you want to select the Current Job, you can specify that only in a pre-built PEX Definition.

ALL jobs is not allowed because the collection processing is much too high for a large number of jobs and can severely impact system performance. Keep the number of jobs low. PEX Analyzer does not tell you how much is too much, but as a rule-of-thumb keep it to less than ten to twenty jobs undergoing collection at the same time.
The phrase “at the same time” implies that collection can be active in a job stream containing more than 10 or 20 jobs in total but less than that are undergoing concurrent PEX Stats data collection. This can happen in some processing runs in which there can be a lot of jobs with the same job name or user ID.

Make sure you understand how many concurrent jobs you want Stats Hierarchical collection to handle and what the effect can be if it is too high. The most significant factor in PEX Stats collection’s effect on performance is the total, for all jobs, of the Call/Return event rate. This is the number of calls and returns per unit time over ALL enabled jobs. If overall system performance degrades and the users complain, the event rate must be reduced by stopping collection, reducing the number of concurrently collected jobs (by changing the PEX Definition or changing the job scheduling), and starting another collection.

**Note:** For more information about Job Selection, see 2.7.6, “Job/Task selection” on page 36.

After choosing Selected jobs, the Add Jobs window in Figure 4-6 appears. This example shows selecting all jobs whose User name is a specific value. Unless you are very familiar with the run time environment naming conventions, be very careful about how many concurrent jobs that generic selection can cause.

Figure 4-7 shows the result of entering a user name and clicking **Add** to enable collection for all jobs with a specific User name.

The job selection window in Figure 4-7 tells PEX to collect data for any job whose user ID matches the input value.
After selecting **Next** after Job Selection is complete, the Summary window in Figure 4-8 appears.

![Collection Wizard - Summary](image)

**Figure 4-8  Collection Wizard: summary of the collection description**

This window shows all of the PEX Definition's values, both the explicitly entered ones and the other's default values, that will be used in the collection.

Selecting **Finish** starts the collection. If scheduling options are specified, they are used; otherwise, collection starts immediately.
4.3 Processing Stats Hierarchical data

Here we discuss processing Stats Hierarchical data.

4.3.1 Coordinating test jobs and collection start/stop

One collection tip is to ensure that Stats collection brackets or encompasses the test job runs. In other words, it is best to start Stats collection before the test jobs start and to end it after they finish.

Following this suggested sequence is not a requirement, but if you do, you can be confident that you collected all activity in the enabled jobs. It ensures that you get the data you asked for.

If the jobs start before PEX Stats or end before PEX Stats ends, the PEX Stats Hierarchical data item called partial count status shows the jobs status as it applies to the completeness of a row's data.

Even though there may be partial data for higher control levels (back towards zero), the data is often adequate for performance analysis. It is possible that there could be some data that does not accurately show everything that happened. The partial count status value of Y means that the row's program name data is accurate even though there's no resource usage accounted for by the program.

4.4 Running Stats Hierarchical data analysis

To run the Stats Hierarchical data analysis after collection finishes, do the following on the PEX Analyzer window shown in Figure 4-9:

1. Select the library (My_colln in this example).
2. Right-click the collection name (Mystatsh in this example).
3. Right-click Analyze Data in the menu.
4. Select Start Analysis Wizard from the sub menu.
5. Select the **Analysis Type** to run, as shown in Figure 4-10. Presently, there is only one option, so select **Next** to continue.

![Figure 4-10  Selecting the type of analysis](image1)

6. To pick the jobs whose data you want to analyze, select **Browse** from the Subset Job menu (Figure 4-11).

![Figure 4-11  Stats Hierarchical analysis: job selection (1 of 3)](image2)
7. As shown in Figure 4-12, select one or more of the jobs to run an analysis on. Click OK to proceed.

The window contains all of the collection jobs. If you expand the window, you see job run time information, such as how much CPU time it used. Also, the jobs list can be sorted by clicking on a column description; left-click to sort in ascending sequence, right-click for descending sequence.

You can analyze multiple jobs. See 4.4.1, “Multiple job analysis selection” on page 100 for more information.

---

**Figure 4-12  Stats Hierarchical analysis: single job selection (2 of 3)**

After selecting which job to analyze and clicking OK, the Subset Job window (Figure 4-13) shows the name of the job/thread to be analyzed.

8. Click Next to proceed.

---

**Figure 4-13  Stats Hierarchical analysis: single job selection (3 of 3)**
The “Summary” menu Figure 4-14 shows the job whose data the Stats Hierarchical analysis will analyze.

9. Click **Finish** to submit the analysis job. That is followed by a confirmation message box (Figure 4-15).

10. Click **OK** to return to the initial submit window.

![Figure 4-14 Stats Hierarchical analysis: analysis job submit](image1)

![Figure 4-15 Stats Hierarchical analysis: job submission confirmation](image2)

c. Select **OK** to return to the collection’s window.

d. Refresh the view and wait for the analysis Status to show Complete and then proceed to view the analysis output (see 5.1, “Viewing Stats Hierarchical data analysis output” on page 106).
4.4.1 Multiple job analysis selection

If the collection contains data from multiple jobs, use the Browse jobs window to select the ones you want to analyze (see Figure 4-12 on page 98). Make your selections and select **OK** to return to the Subset Job menu (Figure 4-16).

Figure 4-17 shows that multiple jobs were selected. PEX Stats submits a separate analysis job for each of them.

![Subset Job window after selecting multiple jobs](image1)

**Figure 4-16** Subset Job window after selecting multiple jobs

![Subset Job selection inset](image2)

**Figure 4-17** Subset Job selection inset

Selecting **Next** takes you to the window shown in Figure 4-18.

![Stats Hierarchical analysis - summary of multiple analysis job submit](image3)

**Figure 4-18** Stats Hierarchical analysis - summary of multiple analysis job submit

The window in Figure 4-18 shows which jobs were selected for analysis.
Select **Finish** to submit the PEX Stats Hierarchical Analysis jobs to the PEX Analyzer job queue (built at install time).

The PEX Analyzer architecture requires the analysis jobs to run one at a time serially. There are data dependencies between the underlying PEX data files that prevent allowing them to run in parallel.

In most cases a PEX Stats analysis run does not take very long (in the tens of seconds or less) to run. Also, there are not very many analysis jobs to run; most of the time there is only one per collection. Subsequent runs of modified SQL analysis run in an iDoctor job thread at i5/OS interactive job priority.

### 4.4.2 Multiple job analyses and member name

Each Stats data view is driven by the PEX Stats Analysis program output files. If there are multiple jobs in a hierarchical collection and you run default analyses over them, each collection's view represents a different analysis output file **member**.

If you look at the members in an analysis output file, you cannot determine which collection job they are related to. The following shows how to determine the relationship.

The collection Mystats has two analyses. The view Subset information at the bottom part of Figure 4-19 shows which collection jobs have been analyzed.
The iDoctor Data Viewer's Open File window in the lower section of Figure 4-20 (entitled Members for selected file:) shows analysis file G_STATSH (the Stats Hierarchical analysis output file) has two members, Q000000002 and Q000000003. The question is which one belongs to a given job's analysis (as shown in the Subset Information in Figure 4-19 on page 101).

The next step is to go to an analysis view (Figure 4-21) and use the iDoctor menus to go to the view's query, analysis, or collection properties.
The query properties, shown in the bottom section of Figure 4-22, show which member the view represents. In this case, it is member Q000000003 in file G_STATSH in library MY_COLLN. The next step is to discover what collection (for example, application) job the data represents.

![Figure 4-22 Collection Properties Query - Report: Analysis Member name](image)

To do that, select OK to return to the data view and use the view menu again to go to the Properties Analysis window (Figure 4-23).

![Figure 4-23 Properties - Analysis - Subset information: job name](image)

In Figure 4-21 on page 102, right-click the view and select Properties → Analysis to bring up the window in Figure 4-23. Here you see the view (the collection data) is from job QZDASOINIT...876329.

**Relate the analysis view for the collected job name**

The preceding discussion describes how you can determine the relationships between the Analysis View, the analysis File Member Name, and the collection data Job Name.

The steps to follow are:

1. Select PEX Analyzer menu → Library name → Collection name → Report Description → data view.
2. Select the file name (G_STATSH in this case) and then the data view.
3. Select Data view → Properties Query → analysis file Member Name.
4. Select Data view → Properties Analysis → collection data Job Name.
Chapter 5. Analyzing PEX Stats Hierarchical data
5.1 Viewing Stats Hierarchical data analysis output

This section discusses analyzing the default Stats Hierarchical data report and introduces some different analysis approaches by modifying the views to accommodate your analysis methodology.

How do you get to the data

Each Stats Hierarchical Analysis job produces a single member in its output file (G_STATSH) in the collection library. Even if there are multiple jobs in the data, each one that has been analyzed has its own separate report. Each report has its own member in the Analysis Report folder. To view the report, select its report description, as shown in Figure 5-1.

![Figure 5-1 Accessing the default Stats Hierarchical report](image)

5.2 The default initial view

Figure 5-2 on page 107 shows a default Stats Hierarchical analysis data view. As with the Stats Flat view, there are right hand columns that are not shown because of space limitations. Use iDoctor’s Record Quick View (see 5.13, “Record Quick View” on page 138) to view all columns and one or more rows at a time.

The PEX Stats Analyzer default hierarchical views show the activity in a program and MI instructions for only one job thread. Analysis of collections that contain multiple jobs or threads generate a separate view (report) for each collection. For more information, see 4.4, “Running Stats Hierarchical data analysis” on page 96.
Chapter 5. Analyzing PEX Stats Hierarchical data

Figure 5-2  The initial Hierarchical report view

This view shows all program and MI complex instruction activity in the order in which it happened. Many rows have a Times Called count greater than one which indicates that a program call using this specific call path configuration occurred multiple times. To do effective analysis of Hierarchical data, you need to understand the Call level, Reference Identifier, and Partial Count Status columns.

### 5.2.1  Call Level, Reference Id, Partial Count Status fields

The default view of the Stats Hierarchical content is similar to the Stats Flat default view. What is different is its organization.

The following fields are used to define the hierarchical view’s structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call Level</td>
<td>The relative (to zero) position of the called program/MI instruction in the job's call stack shows this row's position in the sequence of calls. The lowest numeric value is the highest level of control in the call stack. The highest numeric value is the lowest level of control; it also represents the depth to which the call stack extended. The Call Level can also be considered a last in, first out (LIFO) list. Each row in a data view, with the exception of the first two in which the program names are &quot;**Unknown&quot; or &quot;**LIC&quot;, represents a call to the program or MI instruction in which the call stack is unique.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Call Level</th>
<th>Partial Count Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N</td>
</tr>
<tr>
<td>0</td>
<td><strong>Unknown</strong></td>
</tr>
<tr>
<td>0</td>
<td><strong>Unknown</strong></td>
</tr>
<tr>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>QSYS</td>
</tr>
<tr>
<td>1</td>
<td>QCMD</td>
</tr>
<tr>
<td>1</td>
<td>QCMD</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>QSYS</td>
</tr>
<tr>
<td>2</td>
<td>QCMD</td>
</tr>
<tr>
<td>2</td>
<td>QCMD</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>QSYS</td>
</tr>
<tr>
<td>3</td>
<td>QCMD</td>
</tr>
<tr>
<td>3</td>
<td>QCMD</td>
</tr>
<tr>
<td>4</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>QSYS</td>
</tr>
<tr>
<td>4</td>
<td>QCMD</td>
</tr>
<tr>
<td>4</td>
<td>QCMD</td>
</tr>
<tr>
<td>5</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>QSYS</td>
</tr>
<tr>
<td>5</td>
<td>QCMD</td>
</tr>
<tr>
<td>5</td>
<td>QCMD</td>
</tr>
<tr>
<td>6</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>QSYS</td>
</tr>
<tr>
<td>6</td>
<td>QCMD</td>
</tr>
<tr>
<td>6</td>
<td>QCMD</td>
</tr>
<tr>
<td>7</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>QSYS</td>
</tr>
<tr>
<td>7</td>
<td>QCMD</td>
</tr>
<tr>
<td>7</td>
<td>QCMD</td>
</tr>
<tr>
<td>8</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>QSYS</td>
</tr>
<tr>
<td>8</td>
<td>QCMD</td>
</tr>
<tr>
<td>8</td>
<td>QCMD</td>
</tr>
<tr>
<td>9</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>QSYS</td>
</tr>
<tr>
<td>9</td>
<td>QCMD</td>
</tr>
<tr>
<td>9</td>
<td>QCMD</td>
</tr>
<tr>
<td>10</td>
<td>Y</td>
</tr>
</tbody>
</table>
**Note:** “Unique” means that there is a specific sequence of calling programs that eventually resulted in calling the program represented by the current row. Program X may have been called elsewhere by a different sequence of programs and, if so, there is a different row representing that call to program X.

Figure 5-3 shows the Stats Hierarchical representation of program to program flow, and the relative length of time that each program was “in control”. Call Level is the vertical axis (the dashed line and arrow), time is the horizontal axis (the solid line and arrow). The program in control corresponds to the Call Level value.

![Figure 5-3 Program to Program Calls and Call Level value](image)

The view shows:
1. The i5/OS QCMD program calls the first application program A.
2. A calls program B.
3. B calls C
4. Each of them returns as follows: C returns to B, which returns to A, which returns to QCMD.

Following that, back at Call Level 0:
1. QCMD calls B.
2. B calls C.
3. Each returns as follows: C returns to B, which returns to QCMD.
The inset Call Level . . . . . Program shows the Call Level value for each program as it appears in a PEX Stats Hierarchical analysis default view.

**Reference ID**

Not shown, but very important in the default view in Figure 5-2 on page 107, because it is at the far right of the default data view.

The Reference ID is the “who called whom” tracker. It represents the time order sequence of the first occurrence of each call stack configuration. If there are multiple rows with the same library and called program/module/procedure or MI Complex Instruction, then by definition they have different call stack configurations.

There are two fields that are Reference Id related. One is the Reference ID. Its value is calculated (by PEX Stats collection programs) when a new row is added by the collector. This happens the first time a program is called with a unique call stack.

The second related field is called the **Caller Reference ID**. It contains the Reference ID value of the calling program, that is, the program that called the program shown in the row you are viewing. It is used by the Hierarchical Analysis to build its where used analysis view discussed later.

When a program is called with the same call stack configuration and it repeatedly occurs, its row is the one that receives the accumulated, that is, the Cumulative count and time values for all subsequent Calls that originated from that program.

**Note:** The purpose of Call Level and Reference ID is to preserve the content, sequence, and relationships of calls and returns. They can be used to relate a called program back to the callers. This in turn can infer or show possible causes for the sequence of calls.

Understanding the call stack and reference ID concepts is key to analyzing Stats Hierarchical data’s program to program relationships.

Modifications to the default view often include moving the Reference ID (field REFID) value in the viewable portion of the output.

**Note:** The entries **LIC Task and **Unknown have REFID values of -1 and -2. In almost all cases, a user defined query should exclude the two rows with those REFIDs from a result.
The lower right part of Figure 5-4 shows part of the Stats Hierarchical default view. When each program is called, its Call Level back chain is inspected to see if it is unique. If it is, then a number is assigned to it. This is called its Reference Identifier (REFID).

If Stats collection has already encountered this specific back chain configuration, then no new REFID is assigned; however, B’s Times Called value is incremented by one.

When the new REFID is assigned, another number called the Caller REFID is also assigned that identifies the calling program. This is the calling program’s REFID value.

For example, in Figure 5-4, the Stats Hierarchical partial view shows two occurrences of both program B and program C, each of which has its own unique Call Level back chain:

- The first call to C has a back chain of C → B → A → QCMD and, because it (C) was called by B (which has a REFID of 2), C’s Caller REFID is B’s REFID value (REFID of 2).
- The second occurrence of a call to program C has a back chain of C → B → QCMD and a Caller REFID of 4, which shows that C was called by the second occurrence of B (REFID 4).

Partial Count Status If the value is “Y” the program was:
- Active (had been called) when Stats collection started or
- Had not returned (was still active) when Stats collection ended or
- Both of the above.

A Partial Count Status value of “Y” means that the row’s program activity data is incomplete and accurately represents only the name of the called program and does not represent what resources the row’s program used.
If the value is N, then the program data is complete and can be considered valid for the program/MI instruction.

**Note:** The Unknown row does *not* contain data for programs that are not enabled for collection. The data for non-enabled for collection programs is in the calling program’s row because neither the call to or the return from the unenabled program is detected by PEX data collection, i5/OS STRTRC/ENDTRC, or PEX Call/Return Trace.

**What is the relationship between rows with equal Call Level values**

The following three points assume that the view is in REFID sequence ascending.

Equal Call Level and Intervening Rows: Starting with the current data row’s call level value to the next row that has the *same* call level value, all intervening rows were created by the current row’s program.

For example, in Figure 5-2 on page 107, note the first two rows that are at Call Level 11 (indicated by the arrows).

Between the two Call Level 11 rows, there are three intervening rows at Call Level 12 and 13. These three rows exist (in other words were caused by) only because they were called by the program represented by the first Call Level value 11 row.

**What is the relationship between rows of different call level values**

The second Call Level 11 row is not dependent on the first Call Level 11 row as far as PEX Stats Hierarchical recording is concerned. The second Call Level 11 row exists only because it was called by the program represented by its *closest preceding* Call Level 10 row.

**What is the relationship between cumulative values and call level rows**

Call Levels and Cumulative values: Every row contains both Inline and Cumulative values representing times, counts, and resource use.

*Inline* values represent the resources, times, and counts used within the program or MI instruction only in that row.

Any row’s *cumulative* values are the sum of a row’s *inline* values plus all immediately succeeding rows *inline* values down to, *but not including*, the next row with the same Call Level value as the row where you started.
The Stats Hierarchical Analysis View inset in Figure 5-5 shows what the Stats Hierarchical data would look like if your query selected the fields (columns) for Call Level, Program Name, Inline XXX, and Cumulative XXX. Note the numeric values shown for each of the horizontal arrows in the Call Level diagram. They indicate an amount of time used by each program and are used to illustrate the recording methodology.

There are two types of resource recording, Inline and Cumulative. The inset shows the summed values for the row’s Inline and Cumulative values. In the calculation, the values appear in the time sequence order in which they were generated according to the graphic.

PEX Stats Hierarchical data collection accumulates the data for all resource usage/counts by program-module-procedure by job thread. PEX Stats Flat does the same type of calculation however it does not use Call Levels; it uses only the program-module-procedure or MI Instruction name and job thread.
5.2.2 Viewing the Hierarchical Default Analysis view

The default view is a good starting point if you are familiar with the job you collected data for. In that case, you probably know what program(s) you are interested in and you can find them and start your analysis.

Alternative starting points

If you are not familiar with the test case (jobs and data), you may have to do some searching to find what you are looking for. If so, then you might alternatively start the analysis at either:

- Section 5.5, “Alternative starting point I: Stats Flat view of Hierarchical data” on page 117
- Section 5.9, “Alternative starting point II: a modified default view” on page 127
- Section 5.14, “Alternative starting point III: component resource, counts, and times summary” on page 140

Positioning in the data view

In the default view, you may want to position the data view to a specific program or first occurrence of a user library row to start your analysis. You can do that by:

- Using iDoctor’s Edit → Find menu (there is also an iDoctor icon for it) to bring up a Find menu, fill it in, and go from there. That positions you to the first occurrence of the search argument and selects every occurrence of the search argument. It is not the fastest way to get to the data and requires a specific search argument. If there are a lot of records, Edit → Find can take a noticeable amount of time.

- A faster method is to use iDoctor’s Query Definition → Record Selection tab to define a SQL Select statement’s Where clause to find what you are looking for. For example, the first non-i5/OS library would be the first row whose LIBNAM field does not start with Q. The resulting view shows all occurrences of the search argument.

In either method, you can proceed with analysis. There are at least two options:

1. One option is to do a where used analysis to find the Call Path to the found row from the program(s) that caused it to run,
   a. Right-click the row.
   b. Select Analyze.
   c. Select the type of Where Used analysis from the menu (program, procedure, or MI instruction).

   There is more discussion in 5.3, “Where Used analysis example 1: Ensure Object” on page 114.

2. The other option to see what happened as a result of the program in the “found” row is discussed in 5.11, “Drilling down: what used analysis” on page 130.

   If you started at a MI Instruction, there is no drill down. They do not call anything else, so there is no “what used” data.
5.3 Where Used analysis example 1: Ensure Object

This example starts after finding a row for the Ensure Object MI instruction. This instruction can do a lot of disk writes, object level seize waits, and sometimes induces temporary slowdowns.

Right-clicking a row in a detailed (that is, not a Group By query row) Hierarchical data view brings up the Analyze menu.

Selecting Analyze brings up a sub menu from which you can run a where used analysis. The results of a where used analysis shows what higher (Call) level programs call this row’s program, procedure, or MI Instruction. To run the analysis, select the submenu option that matches which one of the row’s columns you want to use:

- Where used by program
- Where used by procedure
- Where used by MI Instruction

Figure 5-6 shows an example of starting a Where Used analysis request for an *ENSOBJ (Ensure Object) MI Instruction.

![Figure 5-6 Stats Hierarchical Analyze Where Used menu](image-url)
On the next window (the Child Analysis Parameter prompter, which is not shown here), click Analyze to submit the analysis to run in an iDoctor server job thread. Your iDoctor window is disabled while the analysis runs. When the analysis finishes, the child analysis result data view appears (Figure 5-7) and the window is re-enabled.

This view is a new window in the same (numbered) iDoctor Data Viewer used when you selected the analysis (that is, use the Windows command entry on the command bar to rearrange or view the Data Viewer's windows).

**Note:** The Child Analysis output data view is over all columns. You may have modified the data view the request was submitted from; however, the child analysis does not use the submitted menu's view for its output. It always presents all fields in their default order (the order they appear in the G_STATSH file).

![Data Viewer - #1: [RTCALLTEST/Calltestb/Statistics for All Instruction *ENSOBJ, Call levels: 5, Filter percent: 0 - #1](data-viewer.png)](data-viewer.png)

**Figure 5-7  Where Used Example result: Ensure Object instruction**

The Where Used analysis result in Figure 5-7 shows:

- In both cases ENSOBJ was invoked by the program QSYS/QDBDLTFI. The analysis output view contains all Where Used instances of the program or instruction in the original data.
- The intervening rows show that in both cases the ENSOBJs are the result of a File Delete operation (i5/OS program's QSYS/QLIDLOBJ that called QSYS/QDBDLTFI).
- There were two programs that caused ENSOBJ to be invoked even though they did not directly invoke the instruction. Both programs are in user library RTCALLTEST. In the first instance, the responsible user program is CRTPFTEST1; in the second one, it is CRTPFTEST2.

More investigation will show that ENSOBJ does a lot of disk writes. In this example, the disk writes are part of the delete file (or other object) function requires that its disk space be given back to the system.

See 7.3.2, “Disk space usage” on page 165 for more information about ENSOBJ instruction usage.
5.4 Hierarchical analysis tips

This section presents some things to look for in a PEX Stats data view. They are applicable, in some cases, to either Flat or Hierarchical data.

5.4.1 Finding hot spots

There are different approaches that can be taken to analyze Stats Hierarchical data. When you are looking at the data and trying to reach a decision, you may find that there is no right answer as to what a value should be or how much is too much. Sometimes there may be an obvious choice that can be made as to the acceptability of the amount of resources used by a program or a group of related programs.

Sometimes a value has to be evaluated based on its contribution to a larger amount. For example, the amount of time a program spends converting date-time information may be a large part of a job’s run time (not the collection’s run time, since that can exceed the active time of a job you’re interested in).

Another example might involve looking for trade-offs. You can do a hierarchical collection of a job that runs queries and determine its run time. More investigation may show that much of the job’s non-query time was spent in creating data space indexes that were subsequently destroyed when the job finished. In this case, you may have to evaluate the trade-offs between dynamically creating indexes, running the query (that is, scanning) over the whole file, or building a new, permanent index over the file(s). The solution you choose may affect the performance of other jobs that you are not aware of.

In these cases, as well as others, a good decision may involve the evaluation of multiple scenarios until you find what appears to be an optimal solution.

5.4.2 When high direct cost is not what you are looking for

There are instances where you are not looking for the highest cost or count values. Instead, you may find that something occurred that causes other high cost functions to occur within the jobs. The presence of some i5/OS programs (such as QSRSLCPR and other QSR... programs, part of Save/Restore) cause processing in SLIC tasks. Some of these types of programs and MI instructions are discussed, in part, in 3.3.5, “Markers: program name analysis” on page 70 and 3.3.6, “Markers: MI Complex Instructions analysis” on page 72.

5.4.3 What are you looking for

You may be interested in Inline and Cumulative resource usage, for example, CPU, elapsed time, or disk I/O counts. Some counter values may be of interest, such as how many times the program was called, how many calls a program made, the number of MI instructions called, high values in the optional PEX Event counters, and so on.

Any row’s inline value is always less than or equal to its cumulative value. In the case of MI instructions, inline and cumulative values are the same, because MI instructions do not call other programs or instructions.

There are multiple ways to look at the data. You have to decide which of them apply to what you are looking for. You often want to see the collection’s highest cost items, such as:

1. Highest cost program regardless of what job it is in.
2. The top “n” programs in inline costs or counts.
3. The programs in the top “n” percent of inline costs or counts.
Another way of viewing the data may be to see:

1. How often a specific program or MI instruction was called, overall or by job.
2. How often a specific set of programs or MI instructions were called, overall or by job.
3. How much time was spent in the i5/OS Command Analyzer.
4. How much time was spent creating data base indexes.
5. How many times the MI Deactivate Program instruction was used in the collection and furthermore, by which programs.

This instruction itself is not costly, but it has multiple relatively high cost secondary effects on program initialization, file open, and program activation and deactivation, all of which contribute to poor utilization of your system and poor performance.

6. For the total time of a job’s collection, what program(s) make up the major parts of the jobs run time.

7. With the view in REFID sequence (ascending) look at the collection’s Cumulative Elapsed time. Reorder the view by Inline Elapsed time in descending sequence and you will see the major contributors. For an example, see Figure 5-27 on page 133.

8. You might be interested in the cost of full data base Open/Close processing. There is information about this in Figure 5-16 on page 126 for Hierarchical data.

5.5 Alternative starting point I: Stats Flat view of Hierarchical data

If you start your analysis at the Stats Hierarchical analysis data view and you are not familiar with the programs being tested, it may take a lot of time and searching before you find what you are looking for.

Another way to start an analysis of a collection wide hierarchical data view is to sort various columns one column at a time and see what the data is showing you. You may develop some ideas as to the “top 10” rows that need more detailed analysis. This approach eventually works, but it is not the most productive at first while you are learning about the application’s run time characteristics.

In some cases, it may be more productive to first generate a summary view from the hierarchical data’s individual row’s view. This is especially true if there are multiple rows, for example, multiple uses, for program or MI instruction XXX.

An example of this type of view is in Figure 5-8 on page 118. In this view, there is one row per program/module/procedure or MI instruction. The hierarchical data’s call level information does not apply, but there is a new column, the “Count” value, that represents the number of rows in which the program/module/procedure or instruction occurred in the hierarchical view.

The SQL (from Figure 5-9 on page 118) can be changed to select a range of REFIDs so you see only the data you are interested in.
Looking around the data before diving in
The Flat view of the Hierarchical data may provide multiple starting points for an analysis. One way to start is to sort the Flat view by one of the following columns and see what it shows.

5.5.1 Times Called (descending)

This view (Figure 5-8 and Figure 5-9) often shows the high use “callees” to be MI instructions, such as Set Cursor, Retrieve Data Space Entry, Convert Date, Resolve System Pointer, or some of the i5/OS programs, such as the data base program’s QDBGETKY, QDBGETSQ, or others.

The Hier to Flat view also shows how many different places each program or instruction was called from. This is in the “OCCURENCES” column. This implies that there may be many different data base processing functions in the job and most of the processing is single record reads (QDBGETKY and QDBGETSQ) and updates (QDBUDR).

```
SELECT COUNT(CALLCOUNT) AS OCCURENCES, LIBNAM, PGMNAM, SUBSTR(PRCNAM,1,25) AS PROCNAME, MICPXNM, SUM(CALLCOUNT) AS TIMESCALLED, SUM(INELPUS) AS SUM_INELPUS, SUM(INCPUUS) AS SUM_INCPUUS, MAX(INELPUS) AS MAX_INLP_ELP, MAX(INCPUUS) AS MAX_INL_CPU, MAX(CUELPUS) AS MAX_CELP, MAX(CUCPUUS) AS MAX_CCPU, PRCNAM FROM libname/G_STATSH WHERE LIBNAM NOT LIKE '***' GROUP BY LIBNAM, PGMNAM, MICPXNM, PRCNAM ORDER BY TIMESCALLED DESC
```

Figure 5-8  Hier to Flat view: Times Called sort

Figure 5-9  SQL for flat view of hierarchical data (sort key = Times Called descending)
5.5.2 Inline Elapsed Time (descending)

Reordering the previous figure’s data by inline elapsed time shows some of the same entries as the Times Called sort had at the top. There is a new interesting entry, *ESTDSIKR (Estimate Data Space Index Key Range MI Instruction). Its elapsed time per call is relatively high compared to the SETCR and INSDSEN instructions.

ESTDSIKR is used primarily by SQL/Query optimization processing to decide the best way to run the query. If you see it repeatedly, it may be time to review the setup of the SQL and files being used.

![Hier to Flat view: Inline Elapsed Time sort descending](image)

Note the ESTDSIKR instruction’s Elapsed Time (43.582082 seconds) to CPU time (1.157276 seconds) ratio is approximately 43:1. The reason for this is probably due to high synchronous disk I/O counts. Use Record Quick View on this line or scroll to the right to see the disk I/O counts and verify if this assumption is reasonable.

An analyst needs to consider what values can contribute to the Inline Elapsed time. It might include CPU time, DIO time, wait time on locks, seizes, SLIC waits, and so on. It may be necessary to do further analysis with other iDoctor tools, such as Job Watcher or PEX Trace.
5.5.3 MI Instruction (ascending)

Figure 5-11 shows the data sorted by MI Complex Instruction name in ascending sequence.

Evaluation of the use of certain MI instructions is a very powerful but relatively unknown use of PEX Stats, in either the Flat or Hierarchical case. Just the fact that certain instructions occurred is of interest; knowing what application programs caused them is very useful and can often lead you to the source of the problem. No other System i performance tool provides this type of information.

Even though you may not be familiar with the application programs, some of the activity at this level tells you more about what actions the program(s) are taking. In this example:

1. The *DEACTBM indicates that at 23 different places some program(s) returned and deactivated themselves a total of 157 times.
2. *DEACTPG indicates that at 76 different places some program(s) returned and deactivated themselves a total of 89 times.
3. There were 898 *DESCR (Destroy Cursor) instructions from 158 different rows (that is, different Call Stack configurations). This may be related to items 1 and 2 above.
4. *DESDSINX occurred 11 times in six places. This might indicate that an index was built for a query and was no longer needed after the program returned or the job ended. Run a Stats Hierarchical where used analysis to discover more about where it was used.

Investigating the “top 10” programs representing the values shown in other columns is often a productive and efficient way to start a hierarchical data analysis.

5.5.4 Finding the most expensive program

This analysis approach finds the most expensive program’s overall resource usage. Sometimes the highest resource/time value row from a Hierarchical view descending sort may not represent the program that is the most expensive overall.
The data view in Figure 5-12 shows that the first row’s entry for *DESDS (Destroy Data Space) represents two rows (its “Count” column value) from the hierarchical view. In those rows, its maximum Inline CPU time was 20872 microseconds and its total CPU (Sum Inline CPU usec) usage was 25635 microseconds.

However, the most expensive, in terms of overall CPU use, MI Instruction was *CRTS (Create Space) shown in the third row. It was called from 17 different places a total of 1364 times and its largest individual CPU use was 8319 microseconds. However, its total CPU use (Sum Inline CPU usec) was 33190 microseconds, about 15% more than *DESDS used.

In this example, the attention should go to reducing the number of calls to *CRTS, the entry with the highest overall inline CPU cost. Note also that *DESS (Destroy Space) was called the same number of times as *CRTS. That is a strong inference that reducing the CRTS activity will result in a corresponding reduction in *DESS activity.

Restricting the analysis to the default Hierarchical view leads you to work with the *DESDS instruction. Making changes to affect the use of DESDS will likely have a benefit; however, using the Flat, rolled up data view shows potentially more rewarding fixes elsewhere.

As mentioned earlier, you can reorder a view to see the contributors in other categories. For example, this view is in Inline CPU descending order. Reordering it by Inline Elapsed time shows a different set of “top 10” programs.
5.6 Where used analysis: example 2

Here we discuss another where used analysis example.

5.6.1 User defined queries and where used analysis

This section discusses creating a new query over Hierarchical data with iDoctor's New SQL Query window. When you run it the first time and right-click in the data view, the resulting menu does not include the Analyze (where used) menu item. However, after you save the SQL using iDoctor's Query Definition Save As, it is recognized as a PEX Stats User Defined Query (UDQ) and the Analysis menu item is available.

Child analysis tips
As you go through the data, you will see program or instruction rows that have high call counts or high resource usage. Some of them may not seem to fit properly with the application. One example of this “fit” could be frequent use of programs to retrieve job related data, such as QCLRTVJA (Retrieve Job Attributes) or to do date time conversion. If there are frequent uses of programs whose names start with QCL...., such as date conversion, they would run much faster if the more efficient QWC.... program to do the same function were used.

You can run a PEX Stats child analysis over a data view to see a where used view. The purpose of this is to show the call stack from the called program back up through all of the calling programs. Knowing this path can guide you to where changes need to be made.

As an example, the menu settings shown Figure 5-13 on page 123 were used to run a MI Instruction where used child analysis:
1. Right-click the row whose Call Stack back chain (the back chain of calling programs) you want to see.
2. Select Analyze from the menu.
3. Select the appropriate (content sensitive) entry from the submenu. In this case, select Where selected MI instruction used.
4. Submit the analysis.
5.7 Viewing a Where used analysis

This section discusses the results of a where used analysis submitted as shown in Figure 5-13 for MI Instruction *CRTS (Create Space).
Figure 5-14 Child analysis of Hier to Flat view row

This view shows all uses of MI Create Space instruction in the collection. Some are from invoking a CL program, some from create file and some from other causes. Each separate Call Level back chain is represented. You can pick out the most expensive ones and discover where they were called.

The arrows show the cause (left side) is the Create File command's CPP QDBCRTFI; the effect (right side) is the time used by the MI Create Space instruction. The assumption is that most of the elapsed time is due to disk I/O because the Inline CPU time is about 7% of the Inline Elapsed time.

### 5.7.1 Service and non-Service program data recording

This section discusses unenabled ILE program's resource usage recording. It applies to either Service programs or non-Service programs.

Assume a scenario where program A calls B and B calls C and that B is NOT enabled for collection.
In this case, all of the resources, times, and counts that B uses, with the exception of B’s Times called value, are recorded in A’s row. Also, A’s number of calls will not include the calls to B. Furthermore, all of the calls that B makes to C are recorded as calls from A. The bottom line is that as far as the PEX Stats Collection function is concerned, B does not exist, but A and C may have some strange and possibly extraordinarily high numbers. Consider the following from a PEX Stats view:

1. A was called once.
2. C was called 50000 (fifty thousand) times (this is a real example pulled from the archives).

In this example, program C is the data base file Full Open program named QDBOPEN. Given the numbers, the initial conclusion was that A opened 50000 files event though it was called once. However, these are unrealistic numbers. Additional investigation showed that C, the intermediate program, was doing all the opens. You sometimes have to assess how reasonable the numbers are.

An excessive opens and unenabled program example

Figure 5-15 represents part of a situation shown by Stats Hierarchical data from a user who was testing a new application.

<table>
<thead>
<tr>
<th>Call Level</th>
<th>Program Name</th>
<th>Times Called</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JOE_PROGRAM</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>QDMCOPEN</td>
<td>50,000</td>
</tr>
<tr>
<td>3</td>
<td>QDBOPEN</td>
<td>50,000</td>
</tr>
</tbody>
</table>

1. JOE_PROGRAM is a high level language program. When you look at the other entries “times called” data, you immediately know something is not right.
2. What’s the problem? JOE_PROGRAM was called once, but the data shows that the data base file open programs were called 50000 times. That is unreasonable. (We could have picked full file closes and achieved similar results.)
3. After asking a few questions, it was apparent that the program was calling an unenabled program to open the files and perform the processing. The program did not appear in the Stats data
4. There is another point that can raise suspicion. There is a limit to the number of files that a RPG program could have open and that is much less than 50000.

If the program had been properly enabled by using the proper parameters on CRTRPGMOD, it would show up at Call Level 2 and the QDMCOPEN and QDBOPEN programs would each be pushed down by one Call Level.

If you suspect that there are unenabled programs in the call path, try to identify what they are. If you can do a CHGPGM or CHGSRVPGM on them, you can continue your performance analysis work. If not, get their developer to enable them.

For more information, see “Finding a program module’s ENBPFRCOL setting” on page 197.
5.8 Calculating costs of a function

There are some system functions, such as full file open, calling a program, starting a job, processing messages, and so on, that use a specific set of programs to perform it. This example focuses on the costs of full file opens by showing, for each of its parts, its name, the number of times called, total elapsed time, the CPU time used, and the number of unique callers in the collection (or selected range) (see Figure 5-16). These values can be easily accumulated into a cost/per open.

Figure 5-16   Approximate costs of full data base Open/Close

The program and MI instructions names and descriptions are in iDoctor’s QIDRGUÍ/QAPMDESCS file.

Use the SQL shown in Figure 5-17 as a template for other “roll your own” analyses that you may need as you work with Stats Hierarchical data.

The same type of component analysis can be run over Stats Flat data; however, the program causing the problem, may not be apparent. For an example, see the discussion for Figure 3-18 on page 80 in 3.4.2, “Program behavior analysis using Stats Flat Times Called values” on page 79.

```
SELECT PGMNAM, MICPXNM, sum(callcount) AS TOTALCALLS, sum(inelpus) AS TOTINLELP, sum(incpuus) AS TOTINLCPU, count(callcount) AS COUNTCALLERS FROM libname/G_STATSH WHERE PGMNAM LIKE 'QDBOPEN%' OR MICPXNM = '*CRTDOBJ' OR PGMNAM LIKE 'QDBCLOSE' OR PGMNAM = 'QDMCOPEN' OR PGMNAM = 'QDMCLOSE' OR MICPXNM = '*DESS' OR MICPXNM = '*ACTCR' OR MICPXNM = '*DEACTCR' OR MICPXNM = '*DESCR' OR PGMNAM = 'QDMGETOV' GROUP BY PGMNAM, MICPXNM ORDER BY MICPXNM ASC, PGMNAM ASC
```

Figure 5-17   SQL for the DB Open/Close costs query
5.9 Alternative starting point II: a modified default view

Figure 5-18 shows the selection of a different Stats Hierarchical data view with a user defined query

![Figure 5-18](image)

Sometimes there is a need for an alternative view instead of the default analysis view. In this example, the collection’s report folder shows options in addition to the Analysis Reports. One option is to select from the collection’s User-defined-Queries (UDQ). The selected UDQ is **Stats Hier Cum xxx and Refid view**. The selection sub menu shows that you can:

- Open Table(s): Open the view defined by this UDQ (Figure 5-19 on page 128)
- Delete the UDQ.
- Set the scope of the UDQ (collection, library, system, and all systems).
- View its Properties: From there you can copy the SQL and, using the iDoctor New SQL Query icon, paste and modify the SQL and use it instead of the UDQ’s current SQL.

If there is a lot of data or this is a very complex query that takes a while to run and you want to make some changes to it, you could copy and paste it into an iDoctor New SQL Query window and change the query before running it.

See “iDoctor’s New SQL Query view interface” on page 217 for more information.
Figure 5-19  Stats Hierarchical Analysis: user defined report properties (including SQL source)

The SQL source for the view shown in Figure 5-21 on page 129 can be copied from Figure 5-20 when viewing this book online. (It is the same as the User-defined Report Properties shown in Figure 5-19).

See the discussion in “iDoctor’s New SQL Query view interface” on page 217 for a discussion about building and saving your own user defined queries.

Figure 5-20  SQL source for figure Figure 5-21 on page 129

To make it easier to scroll through this data and see the significant values on one window, this view uses fewer fields and a different field arrangement.

For example, the Inline and Cumulative CPU use and Elapsed time values are adjacent. This lets you more easily see the current row’s program or continue to drill down the call stack to the programs called by this row’s program to see the resource usage values.

This view shows the Call Level, library, procedure, MI instruction, Times Called, Inline and Cumulative Elapsed times and CPU usage, Total Synchronous and Asynchronous disk I/O counts, and Reference ID.
5.10 Summarized Hierarchical view

In this example, the Call Level 7 rows are highlighted. In the second highlighted row, the Elapsed time values are about 15% of the Cumulative values.

You want to view all rows related to the second highlighted Call Level 7 row. The view has to be sorted in REFID ascending sequence.

To put it another way, you want to view all the rows starting at the selected Call Level 7 row and going forward to, but not including, the next Call Level 7 row.

The easiest way to do this is to change the view (SQL) to show only those rows. Do this by using iDoctor’s Query Definition Record Selection - Range selection on the REFID field.

You want to view the rows whose REFID fields are in the range 67 through 78 inclusive. This may look like a contrived example, and to keep it manageable here, it is. In the real world, you will encounter hundreds or thousands of rows that you need extract information about to find the high cost ones.

What you are looking for in the REFID range from 67 to 78 are the rows with the highest Inline CPU or Elapsed time value, because these are the biggest individual contributors to the resource usage or run time; this analysis is not interested in the lower level row’s Cumulative values.

In this example, the RSLVSP instruction at Refid 75 is the largest contributor to the Call Level 7 row’s (REFID 67) cumulative times. This is found by first determining the range of rows of interest and then finding the highest Inline Elapsed time row (for example, do a descending sort on that column).
PEX Stats Hierarchical where used analysis is good for drilling up, that is, show who called a given row, and it applies to many analysis situations, such as when you have a list of candidates, such as CRTDOBJ (representing full file open) or ENSOBJ (indicating a high incidence of synchronous disk writes).

5.11 Drilling down: what used analysis

This section discusses how to find the programs that contributed the most to an application’s higher level calling program’s cumulative values (or what programs it did call). In other words, find the called rows with the highest inline values. The first step is to find all the programs that the higher level program called or caused to be called, that is, everything down the call stack).

To do this step by step:
1. If you are hunting and just looking for application program data, start by finding the first call to a Program in a user library, that is, a library whose name does not start with “Q”.

   The first row that meets this criteria is REFID 61 in Figure 5-21 on page 129.
2. Record that row’s Call Level (field CALLVL) value (6 in this case).
3. Record its Reference ID (field REFID) value (61 in this case).
4. Starting at the row where you found the first program in step 1 above, find the next row with the same call level value (6) and record its REFID value minus one. This represents the last row related to the (starting) record found in step 1 above.
5. Everything the starting row caused is in the REFID range starting with what you recorded in step 3 and ending with what you recorded in step 4.

   This is shown in Figure 5-22, which built by the SQL in Figure 5-23.

5.11.1 Analyzing the REFID range

Figure 5-22 Application program call level REFID range

```
SELECT CALLVL, REFID, LIBNAM, PGMNAM, MICPXNM FROM libname/G_STATSH WHERE CALLVL = 6 ORDER BY REFID ASC
```

Figure 5-23 SQL for the view in Figure 5-22
The results in Figure 5-22 on page 130 show that the first non-Q library program is RTCALLTEST, which appeared at Call Level 6 and has a REFID value of 61.

The next Call Level 6 entry is at REFID 633. Given the methodology used to generate the REFID value, it is valid to conclude that all program activity from REFID 61 to 632 (inclusive) was caused by the program CALLTEST. Based on this conclusion, we can focus on all the work that occurred within that REFID range.

Note: In this example, if a succeeding Call Level 6 record was not found (633 in this case), it means that either the program did not finish before the collection ended or that it was the last Call Level 6 program and the run ended after it finished. If it is the first case, the data may not be accurate and the test should be rerun. Check the data for “reasonableness”.

The next step is to change the SQL so the view shows the Call Level 6 starting row and each of the Call Level 7 entry's data within CALLTEST's REFID range (see Figure 5-24).

```
SELECT CALLLVL, LIBNAM, PGMNAM, PRCNAM, MICPXNM, CALLCOUNT, INELPUS, CUELPUSS, INCPUUS, CUCPUUS, (INSDR+INSNR+INSDW+INSNW) AS SDIO, (CUSDR+CUSNR+CUSDW+CUSNW) AS ADIO, REFID FROM libname/G_STATSH WHERE REFID BETWEEN 61 AND 632 ORDER BY REFID ASC
```

Figure 5-24   SQL for the view in Figure 5-25

The view in Figure 5-25 shows:

1. The total elapsed time for program CALLTEST and everything it called is 2.103510 seconds (first row Cumulative Elapsed us column).

2. There are two major contributors to this elapsed time:
   a. The largest contributor is program CRTPFTEST2 that started at REFID 430. That program and everything it called took 0.950826 seconds (Cumulative Elapsed us).
   b. The second largest contributor is program CRTPFTEST1 that started at REFID 238. That program and everything it called took 0.816025 seconds.

In this example, we analyze the larger contributor of these two (in the real world, you may need to look at a number of them). To do this requires viewing all the activity from REFID 430 to 621. That REFID range has all the activity caused by program CRTPFTEST2.
The expanded view in Figure 5-26 contains 192 data rows, all but the first being direct descendents of program CRTPFTEST2. Some of the rows contain very low cumulative elapsed time values; some in the low microseconds. Other rows have larger values.

One assertion that can be made is that the very low cumulative values can be excluded from this analysis because their elapsed time contribution is relatively insignificant.

In this view, the query is changed to:

- Select records in the REFID range of 430 to 621.
- Select records whose Cumulative Elapsed time value is ten milliseconds or larger,
- Sort it descending by Cumulative Elapsed time.
- Add in the MI Complex instructions field (column) that was omitted from the previous views.

This results in the view in Figure 5-27 on page 133.

By omitting the low value Cumulative Elapsed time records, the amount of data goes from 202 records to 19 records. This makes it much easier to find results when the insignificant values are discarded.
Chapter 5. Analyzing PEX Stats Hierarchical data

To put the data back into a Call/Return perspective, click the Reference ID column header. The result in Figure 5-28 shows the calling sequence and high contributors in each call path.

This view provides insight into the range and distribution of the Cumulative times and highlights which program or MI instructions are involved.
Look at the Inline Elapsed us values in the selected rows. It shows that the largest single contributors to CRTPFTEST2's run time were the MI Create and Destroy Space instructions.

You might ask, why not just do a descending sort on Inline Elapsed time over the whole set of data and analyze that? In some situations, that is the right approach. However, using this alternative in which the view is bounded by a range of Reference IDs shows the activity within specific parts of the application. In other words, what was user program XXXX doing that took so long? That is not apparent from a where used (going up the call stack) view and can only be determined from a what used (who/what did program XXX call?) view.

In the window shown in Figure 5-28 on page 133, do a PEX Stats Hierarchical where used analysis to find the full path down to here and for all other uses of the instruction in the data. To do that:

1. Right-click the row that has the program/MI instruction you want to analyze (*CRTDS in this example).
2. Select Analyze to get the submenu.
3. There are three options depending on what you are analyzing:
   a. Where selected program used
   b. Where selected procedure used
   c. Where selected MI instruction used
4. In this case, we select Where selected MI instruction used because we want to see the call path to the Create Data Space (*CRTDS) instructions (see Figure 5-29 on page 135).

In this example, this analysis step is not necessarily needed, because all the data is in one window. In a real world situation, there can be thousands of data rows to view and the only reasonably efficient way to do it is with PEX Stats where used analysis.
Use the menu defaults and select **Analyze** to submit the analysis run (see Figure 5-30).

The *NO default on “Submit as batch job?” runs the analysis as a thread of the iDoctor server job, that is, at a relatively high priority. If you enter *YES, it is submitted to the PEX Analyzer JOBQ in the batch subsystem. Its run time will depend on how busy the system is.

The Where Used analysis view of the CRTDS instruction shows the call path back, one level at a time (see Figure 5-31 on page 136).
It shows all instances of CRTDS. Make sure that you identify which of these are in the REFID range that you originally selected and you have been working with.

They can and often are, called from different application programs.

Not all intermediate rows are in this view. In this example, not all REFIDs are shown. What is shown is the Call Level path back to the lowest numeric value.

The analysis view shows all input file columns. This example uses a modified view showing only a few columns.

In your analysis, you need to consider what values can contribute to the Inline Elapsed time. This can include CPU time, DIO time, wait time on locks, seizes, gates, and so on. It may be necessary to do further analysis with other iDoctor tools, such as Job Watcher or PEX Trace.

Why do an equal number of calls of the second CRTDS nearly double the elapsed time cost of the first one?

In this example, the second Create File Member (the i5/OS Data Base File create member program QDBCRTME, which invoked the *CRTDS instruction) used a different parameter setting that is controlled by the user CL specifications.

The first data base file member create used the default space allocation setting, the second one used CRTPF ALLOCATE(*YES), which directs the system to pre-allocate, at creation time, all of the disk space that the file will need. While pre-allocation slows down the create (and delete as well), it significantly speeds up file added record processing. When space pre-allocation is used, the application jobs never have to wait for more space to be added to the file which, when there are a high number of concurrent users, can cause noticeable slowdowns.

### 5.12 Where are the Child Analysis Reports

A child analysis is a subset of an existing analysis.

All child analysis reports are stored in the collection’s Child analysis reports Report folder (see Figure 5-32 on page 137). Double clicking it brings up the list of stored reports.
(Figure 5-33 on page 137). You select from the list to view previously run analyses; there are menu options to delete them when you are through with them.
5.13 Record Quick View

Figure 5-34 shows the child analysis for MI Instruction *CRTDS from the Report descriptions in Figure 5-33 on page 137.

![Figure 5-34 Getting to an iDoctor views Record Quick View](image)

iDoctor's Record Quick View shows the data from one or more rows. You can compare the data between nonadjacent rows. For example, to view the two instances of *CRTDS shown in Figure 5-34, do the following:

1. Select the rows to view using Ctrl-click each one.
2. Right-click one of the selected rows to get a menu.
3. Select Record Quick View from the menu.

If the menu's Record Quick View is grayed out and cannot be used, re-enable it by:

1. Double-click one row of the view to bring up the Record Quick View for one record.
2. Select the check box Allow multiple records in the window's upper left hand section.
3. Click OK to exit.
4. Repeat steps 1, 2, and 3 above.
This shows a window similar to Figure 5-35 with multiple data columns in it (one per selected row).

Figure 5-35 is a window similar to Figure 5-35 with multiple data columns in it (one per selected row).

![Figure 5-35: Record quick view results comparison](image)

Figure 5-35 is an iDoctor Record Quick View of the previous data view’s selected records. Using this is one way to visually compare the data between records.

The reasons for the performance differences for the two *CRTDS uses in this example appear to be the number of disk I/O operations and, to a lesser extent, the CPU time used by each. The data shows:

1. Record 10 had four more synchronous non-data base read operations.
2. Record 10 had 269 more synchronous non-data base write operations, probably the most significant contributor.
3. Record 10 had 7 more asynchronous data base write operations.
4. Record 10 used about 8 millisecond more CPU time than record 6.

In this test, the times called value was the same for each comparison row. This may not always be the case and you should normalize the data to a per invocation amount prior to comparing them.
5.14 Alternative starting point III: component resource, counts, and times summary

Depending on the situation, this analysis could be another place to start a Hierarchical analysis.

The amount of resources used by component, such as a group of related i5/OS programs or MI instructions, may be of interest especially when comparing the results of multiple tests. Often a component may consist of program names that have the same characters in position two and three. MI instructions can be grouped by the second, third, and fourth positions of the instruction name.

For example, Figure 5-36 shows the component, its inline CPU time, its total inline synchronous and asynchronous disk I/O requests, its total inline elapsed time in microseconds, the number of times it was called, the number of different callers, and the beginning and ending rows (REFIDs) for each component in the data for different sets of related i5/OS programs and MI Instructions.

![Figure 5-36 Component CPU and times called summary](image)

Some of the i5/OS Component sets shown in the OS Component column are:

- **CA**: Command Analyzer
- **DB**: Data Base
- **MH**: Message Handler
- **CL**: CL Program handler
- **DM**: Data Management

Some of the MI Complex instruction column components are:

- **DES**: Destroy (segment)
- **CRT**: Create (segment)
- **RSL**: Resolve (pointer)
- **MAT**: Materialize (look at object properties)
- **SND**: Send (message, exception, ...)
- **ACT**: Activate (cursor)
- **FND**: Find (index,...)
- **EST**: Estimate (used by SQL)
The view was built by the SQL shown in Figure 5-37.

If the installation you are working with has program naming standards, it might be straightforward to add Record Selection and Group By criteria for application programs.

```
SELECT substr(pgmnam,2,2) AS COMPONENT, substr(micpxnm,2,3) AS MICOMP, 
sum(incpuus) AS CPU_USEC, sum(insdr+insnr+insdw+insnw) AS TOTSDIO, 
sum(inadr+inanr+inadw+inanw) AS TOTADIO, sum(inelpus) AS TOTALINELPUS, 
sum(callcount) AS CALL_COUNT, count(pgmnam) AS USERCOUNT, min(refid) AS 
STARTREFID, max(refid) AS ENDRFID, max(refid) - min(refid) + 1 AS REFIDRANGE 
FROM libname/G_STATSH WHERE (LIBNAM LIKE 'Q%' AND PGMNAM LIKE 'Q%') OR MICPXNM 
LIKE '*%' AND REFID >= 0 GROUP BY substr(pgmnam,2,2), substr(micpxnm,2,3) ORDER 
BY CPU_USEC DESC
```

Figure 5-37   SQL for the components view

**What next with this data**

For example, as a follow on analysis, take a more in-depth look at the data base programs (the DB component that is selected in the figure).

To do that, build a UDO to select all programs whose names start with QDB and are in the reference ID (REFID field) range 116 to 573. With that view, you can do a PEX Stats where used analysis to determine more about what program(s) are causing the high data base use.

One productivity feature of this type of “component” analysis is that it selects collection information based on the collected data. You do not have to search line by line through a large analysis view to determine what programs and MI instructions were used; instead, you can decide to select Data Base, SQL, Query, Save/Restore, Command Analyzer, or some combination of them. The summary shows you where to start drilling down into the detailed analysis.

You could apply the concepts in the analysis discussed in 6.2, “Setting priorities: using the 80-20 rule” on page 149 to select only those components that reach a certain level of resource usage.
Additional PEX Stats data analysis topics

This chapter discusses some additional PEX Stats Analysis, iDoctor SQL and Viewer functions, operating system or other Component's resource usage and distribution, and other information that can help you be more productive in your performance analysis work.
6.1 Resource Usage by Component code: Flat by Pgm data

When an application is running, much of the work performed and resources consumed are done in i5/OS MI programs and MI Instruction related SLIC programs. Sometimes it is useful to know how much activity there is in various components.

6.1.1 Application component cost data

An application’s system components can be identified by using the second and third characters of an i5/OS program (all of which start with Q and are in library QSYS (or some other library whose name starts with Q) and by the first three characters of a MI Complex Instruction (they all start with an asterisk (*) and have no library name).

A component cost graph of this type of data organization shows the CPU used by the programs and MI instruction components (see Figure 6-1).

The graph’s primary Y axis shows that the MI SET... instructions had the highest CPU Use followed by the i5/OS programs QDB.... (data base) and MI CVT... (Convert time or data).

The secondary Y axis shows the highest Total Disk I/O counts occurred in QDB... also (no surprise there) and then in MI CRT... (Create some object) and MI DES... (Destroy some object). This result indicates that the create/destroy activity might be higher than expected and should be investigated in more detail.
6.1.2 Building the SQL

Each operating system program and MI Complex instruction can be considered to belong to one of a number of different components. For example, QDBOPEN and QDBGET and all other programs in the QSYS library whose names start with QDB are part of the data base component. Because the name of each i/OS program in QSYS starts with a Q, this approach uses the second and third characters to classify the programs.

Similarly, all MI complex instructions whose names start with RET, such as RETDSEN and RETSDSE, are part of the SLIC data base Retrieve data records component. All the MI Complex instruction names start with an asterisk, which can be ignored, and the classification can be based on the second, third, and fourth characters of the name.

Figure 6-2 shows the component resource use and elapsed times. The first row, for MI Instruction component SET, shows the overhead for the data base Set Cursor instruction for the whole collection. The second row shows the same type of data for the program component DB, the i5/OS data base component.

The SQL in Figure 6-3 on page 146 generated the view in Figure 6-2.

6.1.3 What does this view show

▶ There are 174 rows in this report, as shown by the count in the lower right hand corner.
▶ The program component's (PGMCOMPONENT) column in the higher cost part (the sort key is Total CPU descending) of the view are:

- DB: Data base.
- JO: Journaling.
- WC: Work Control.
- LE: ILE run time support for application programs.
- DM: Data Management (for example, common functions such as the Open Router).
Some of the MI instruction component's (MICOMP) column contents are:

- **SET**  
  Data base Set Cursor: file positioning.

- **CVT**  
  Convert, often time and date conversion.

- **RET**  
  Data base record retrieval.

- **RSL**  
  Object management's Resolve Pointer. These happen a lot in almost all environments but are usually low cost.

- **MAT**  
  Materialize object: Used to view/retrieve a MI object's attributes/properties.

- **CRT**  
  Object creation: Auxiliary Storage Management (ASM) building an object and allocating space.

- **DES**  
  Destroy object: ASM again to return space.

### 6.1.4 SQL for component classification

The SQL for component classification is shown in Figure 6-3.

```sql
SELECT substr(pgmnam,2,2) AS PGMCOMPONENT, substr(micpxnm,2,3) AS MICOMP,
       sum(qstinv01) AS TIMESCALLED, sum(inlelp01) AS TOTALELP, sum(inlcpu01) AS TOTALCPU,
       sum(qiisdr01+qiisnr01+qiisdw01+qiianr01+qiianw01) AS TOTDIO, sum(qiisdr01+qiisnr01+qiianr01) AS TOTDIOR,
       sum(qiisdw01+qiisnw01+qiianw01) AS TOTDIOW, max(qstinv05) AS THRDCOUNT
FROM /G_STATSFPG
WHERE (MICPXNM LIKE '%%' OR LIBNAM LIKE 'Q%') GROUP BY substr(pgmnam,2,2),
       substr(micpxnm,2,3)
ORDER BY TOTALCPU DESC
```

*Figure 6-3  SQL for the Component cost summary view*
6.1.5 Removing lower cost items from the view

You may want to view, or more likely graph, only part of the data from Figure 6-2 on page 145. Many rows have very low data values and often are not of interest. To exclude the rows that have low costs, modify the SQL to filter them out.

The original view had 174 rows, too many for graphing. The modified SQL's view has only fourteen data rows. Adding a HAVING clause to the SELECT statement's GROUP BY section includes only components that exceed 18489999 microseconds total CPU time. This is the having sum(inlcpu01) > 18489999 clause highlighted in the SQL Query View window's top section (Figure 6-4).

![Figure 6-4 Modifying the view to exclude lower cost items](image-url)
6.1.6 Graphing the component cost data

To graph the data using iDoctor's User Defined Graphs, right-click the data view, select User Defined Graphs from the menu, and proceed from there.

A graph view of the tables can enhance the results presentation to others, especially the decision makers (see Figure 6-5).

For more information about building your own graphs, see “Graph Definition interface” on page 219.

![Operating System Program and MI Component cost summary Graph](image)

This is the same graph used in the beginning of this chapter. Sometimes a graph provides an easier way to interpret the results. In an iDoctor graph, the optional legend on the right shows the graph’s properties, which include:

- **X-axis labels**: This graph used two sets of labels: one to identify the Program Component, the other to identify the MI Instruction component.
- **Primary Y axis**: Each component’s total CPU used.
- **Secondary Y axis**: Each component’s total disk I/O count.

The data shows the CPU and disk I/O values are relatively high for MI SET... (data base Set Cursor); this is expected in data base processing.

Even though their CPU time is much less, the MI CRT and DES (create and destroy) components are of special interest. They both have higher than average CPU use and high Disk I/O counts relative to most of the data. They should be investigated in more detail.

What use is this view? It provides a high level summary of where the highest cost component use is. In some cases, the individual program, job, or MI instruction with the highest resource usage is not necessarily the end of the analysis. Grouping the costs by other criteria, such as Data Base, Save Restore, Auxiliary Storage Management, Command Analyzer, or other components may show much higher than expected costs. Drilling down into that component...
may provide useful results by identifying higher cost programs that were not initially considered. This approach can apply to much of the i5 performance data, not just to PEX Stats data.

Drilling down into components to find more detail may reveal programs and instructions whose high values make them candidates for change. On the other hand, you may find that resource use is evenly distributed over many component members and other performance improvement options should be explored.

It can be useful in studying costs to consider a statistical value called the Coefficient of Variation (COV). It is calculated by dividing the data’s standard deviation by its mean (average). Its value indicates the distribution of the variation in the data’s values. A COV that is very close to one indicates that the data values have very little, if any, variation and further search for value peaks and valleys will not be productive. A higher COV (possibly greater than three or more) indicates wider variations in the data values and more investigation may be useful.

How would you drill down? In this Stats Flat data example, to look in more detail at create and destroy use, change the SQL to show all MI instructions that start with CRT or DES to discover what jobs issued them. Use the job identification data for a follow on Stats Hierarchical collection that also collects the optional PEX ASM event counts to find the programs causing the create and destroy traffic.

### 6.2 Setting priorities: using the 80-20 rule

In many disciplines, including computer performance data analysis, the data often shows that about 80 percent of the activity (or resource use or other costs) is caused by about 20 percent of the sample population. Sometimes this is referred to as the 80-20 rule.

The user often wants a prioritized list of what needs to be changed so they can get the best return on their performance improvement investment. The challenge to the analyst is to provide a prioritized list of the big hitters; the largest resource and time consuming programs that need to be studied. Applying the 80-20 rule approach facilitates that task.

This example shows how to calculate the percent and cumulative percent distribution of a Stats Flat collection’s Inline CPU use by programs and MI Instructions. A SQL sub-SELECT calculates and sums the percentage values. Other variables can be studied by modifying the SQL column selection.
Figure 6-6 shows the cumulative percent distribution of the program and MI instruction Inline CPU usage (in descending sequence). Each row's CUM_PCT column contains the cumulative sum of the current and all preceding row's INL_PCT (inline CPU use percent of the overall use).

The 26th row (see the arrows) is where the cumulative percent reached the 80% level. This example has 212 rows so about 12% of the entries represent 80% of the CPU usage.

6.2.1 Other observations

1. Inspect the smaller value rows. They may be “markers” of other usage.
2. Most of the first 26 contributor’s usage was in the one to three percent range. This provides somewhat of a guideline, at least for this data, as to the magnitude of the important values.
3. In this data, at the 80% usage level, the individual row contribution dropped below one percent.

6.2.2 Graphing the 80-20 data

The vertical bar graph shows the cumulative percentage of the program’s and instruction’s contributions.

The line graph (the 2nd Y axis) shows the value for each row’s inline CPU Percent.

The graph can be built by right-clicking the table view and selecting Graph Definition.
Chapter 6. Additional PEX Stats data analysis topics

6.3 Cost components in Hierarchical data: travel time

Note: This discussion assumes that you're familiar with the discussion of Call Level and Reference ID in 5.2, "The default initial view" on page 106.

This example shows you how to determine indirect costs, for example, the cost of getting there and the cost of getting back, associated with doing a function.

Most of the time, Stats data analysis focuses on programs that have high values for elapsed time, call counts, resource usage, or disk I/O counts. The programs that have these characteristics get a lot of attention in an attempt to reduce their performance impact. This approach is often worthwhile; however, sometimes there may be additional impact due to indirect but related activity.

In the general case of a call from program A to program B, there are costs, such as the amount of elapsed time, CPU time, and disk I/O operations, to perform the call.
These costs can be seen in this example using Stats Hierarchical data to show the processing required to perform a Command Language (CL) program call. Figure 6-8 shows program CALLTEST (at Reference ID 60, the first row) at Call Level 6 calling a number of other programs in library RTCALLTEST. In the test case, program CALLTEST called programs CALLTEST2, DBOPEN, DBOPEN2, CRTPFTEST1, CRTPFTEST2, and the i5/OS Command RTVJOBA (Retrieve Job Attributes).

This example looks at the cost required to call the RTVJOBA Command Processing Program (CPP), which could be thought of as the travel cost, and compares it to the cost within the RTVJOBA program as it is doing its job (which could be considered as the function cost).

![Stats Hier: cost of getting there and back versus the cost of the function](Figure 6-8)

### 6.3.1 Interpreting the data

1. Figure 6-8 shows part of the collection’s data, that is, the Stats Hierarchical data for calling the Retrieve Job Attribute’s function multiple times from a CL program.

2. It shows that at REFID 679 program QCLRTVJA was called and it ran at Call Level 7. Because it ran at Call Level 7, it is valid to assume that it was called by a Call Level 6 program.

3. The data at REFID 664 shows the OS program QCADRV (also known as the Command Analyzer Driver) running at Call Level 7. In this case, it is the first in a series of programs used to handle CALLTEST’s request to run QCLRTVJA.

4. QCADRV does some work, returns, and another i5/OS Command Analyzer program, QCATRS, runs at Reference ID 677. After it finishes, the preparatory work is complete and program QCLRTVJA performs the Retrieve Job Attribute’s function.

5. The RTVJOBA finishes at Reference ID 694 (not shown, but it is the REFID right before the Level 6 at REFID 695).

6. Control passes back to CALLTEST. It is finished and does an ENDPGM and returns to the Call Level 5 program that invoked it.

There are some calculations needed from the data from Figure 6-8 to get the comparative cost numbers. This example uses Excel® to do the calculations.
To get the data into Excel from an iDoctor Data Viewer, use the following steps:

First, save the views data onto the PC. From the iDoctor data view, start by selecting **File → Save → View As...** (Figure 6-9).

Selecting **View As...** brings up the Save As menu (Figure 6-10).

Set the file name and save it as a .CSV (Comma Separated Variable) file on your PC. If you are using some other program and file extension name, and if that extension is supported by iDoctor, use it instead of CSV.

iDoctor can save a view as either .CSV, .TXT, or RTF format.
Start Excel and load or import the CSV file. The result is similar to Figure 6-11.

In this example, there are some blank lines added to delimit different sections.

After the file is loaded, do the calculation and annotation you want in the final result.

<table>
<thead>
<tr>
<th>Call Level</th>
<th>Library Name</th>
<th>Program Name</th>
<th>Procedure Name</th>
<th>Inline Elapsed us</th>
<th>Cumulative Elapsed us</th>
<th>Inline CPU us</th>
<th>Cumulative CPU us</th>
<th>Reference ID for Pgm/Mod/</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>RTCALLTEST</td>
<td>CALLTEST</td>
<td>_CL_PEP</td>
<td>107906</td>
<td>4306231</td>
<td>45291</td>
<td>1054170</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>QSYS</td>
<td>QCLRSLV</td>
<td>QCLRSLV</td>
<td>34</td>
<td>222</td>
<td>10</td>
<td>114</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>QSYS</td>
<td>QCLLCPR</td>
<td>QCLLCPR</td>
<td>7993</td>
<td>42197</td>
<td>3006</td>
<td>192588</td>
<td>66</td>
</tr>
<tr>
<td>7</td>
<td>RTCALLTEST</td>
<td>CALLTEST2</td>
<td>_CL_PEP</td>
<td>7456</td>
<td>19809</td>
<td>4274</td>
<td>9024</td>
<td>78</td>
</tr>
<tr>
<td>7</td>
<td>RTCALLTEST</td>
<td>DBOPEN</td>
<td>DBOPEN</td>
<td>3540</td>
<td>104060</td>
<td>1367</td>
<td>41450</td>
<td>91</td>
</tr>
<tr>
<td>7</td>
<td>RTCALLTEST</td>
<td>DBOPEN2</td>
<td>DBOPEN2</td>
<td>5782</td>
<td>193322</td>
<td>2264</td>
<td>77876</td>
<td>188</td>
</tr>
<tr>
<td>7</td>
<td>RTCALLTEST</td>
<td>CRTPFTEST1</td>
<td>_CL_PEP</td>
<td>8578</td>
<td>1204686</td>
<td>542</td>
<td>61737</td>
<td>282</td>
</tr>
<tr>
<td>7</td>
<td>RTCALLTEST</td>
<td>CRTPFTEST2</td>
<td>_CL_PEP</td>
<td>1066</td>
<td>1181959</td>
<td>541</td>
<td>88363</td>
<td>473</td>
</tr>
<tr>
<td>7</td>
<td>QSYS</td>
<td>QCADRIV</td>
<td>QCADRIV</td>
<td>182128</td>
<td>1179564</td>
<td>66437</td>
<td>604985</td>
<td>664</td>
</tr>
<tr>
<td>7</td>
<td>QSYS</td>
<td>QCATRS</td>
<td>QCATRS</td>
<td>92602</td>
<td>105201</td>
<td>48855</td>
<td>49710</td>
<td>677</td>
</tr>
<tr>
<td>7</td>
<td>QSYS</td>
<td>QCLRTVJA</td>
<td>QCLRTVJA</td>
<td>97199</td>
<td>167135</td>
<td>34416</td>
<td>56331</td>
<td>679</td>
</tr>
<tr>
<td>7</td>
<td>QSYS</td>
<td>QCLRTNE</td>
<td>QCLRTNE</td>
<td>12</td>
<td>104</td>
<td>1</td>
<td>18</td>
<td>683</td>
</tr>
<tr>
<td>7</td>
<td>QSYS</td>
<td>QCLLCNUP</td>
<td>QCLLCNUP</td>
<td>19</td>
<td>41</td>
<td>3</td>
<td>6</td>
<td>691</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>695</td>
</tr>
</tbody>
</table>

| Sum        |              |              |                | 1284765           | 654695            |              |                  |                          |

| Ratio      | 7.7          | 11.6         |

Figure 6-11  The Numbers: Cost of getting there and back versus cost of the function

6.3.2  What costs are of interest

Analysis shows that there are programs running at level 7, 8, and 9 before Retrieve Job Attributes is run.

1. Figure 6-11 shows the total Cumulative CPU and Elapsed Time between starting the Command Analyzer Driver (right hand column: at REFID 664) and finally giving control to QCLRTVJOBA (at REFID 679). These are the sums of the Cumulative Elapsed and CPU us columns for program QCADRIV and QCATRS.

2. The total Inline Elapsed us value for the startup program’s QCADRIV and QCATRS was 1,284,778 microseconds, and the total Inline CPU us was 654,688 microseconds.

3. The total Inline Elapsed us value for the RTVJOBA program QCLRTVJA was 167,135 microseconds, and the total Inline CPU us was 56,331 microseconds.

6.3.3  Conclusions

1. The travel time (in this case, the cost ratio) to get there was 7.7 times longer than it took to perform the function.

2. The CPU time cost to get there was 11.6 times the CPU time used in the function.

These ratios show the importance of knowing the overhead cost as well as the function cost.
The SQL for this example is shown in Figure 6-12.

```sql
SELECT CALLLVL, LIBNAM, PGMNAM, MICPXNM, CALLCOUNT, CALLMADE, CALLMICPX,
INELPUS, INCPUUS, CUELPUS, CUCPUUS, REFID FROM libname/G_STATSH WHERE REFID BETWEEN 60 AND 695 ORDER BY REFID ASC
```

*Figure 6-12  SQL for the getting there and back cost example*

**6.3.4 Summary**

The example shows that the resource and time costs for starting and stopping a program need to be considered in addition to the costs within the function. Sometimes the cost of getting there and back is much higher than the function cost. This is not always obvious on a Stats Flat report or when using the more common approaches to analyzing a Stats Hierarchical report. Often the analyst focuses on the particular function and not on the infrastructure costs surrounding it.

*Note: In this type of example, the CPU time values probably will be consistent from one test run to the next. However, the elapsed time values may vary a lot as the job contends for CPU, main storage, and disk I/O access.*

**6.4 Using Stats to find optimum test case performance**

This example shows the use of PEX Stats Flat to measure the performance of a set of programs that each perform a number of Add operations to a variable. Both ILE and OPM programs are used, each program has a different data type (Integer, Packed Decimal or Binary), and different data lengths and both optimized and unoptimized programs are used.

The test data shows the program name and its Inline CPU time.

There are four “base” programs used to establish the CPU overhead of calling the program and going through its DO loop. The difference between the base programs and the test programs is that the “ADD” instruction is omitted (Figure 6-14 on page 156 and Figure 6-15 on page 156). The base program overhead is subtracted from the test programs values, leaving a result that is assumed to represent the Add operation's CPU time. They perform two sets of non-arithmetic operations. They initialize the variable and execute the DO loop (in this case they initialize, increment, test for a limit, and branch back to the start of the loop).

Figure 6-13 shows the SQL for view the Add Times test results.

```sql
SELECT PGMNAM, INLCPU01 FROM libname/G_STATSFPG WHERE PGMNAM LIKE 'ILE%' OR PGMNAM LIKE 'OPM%' ORDER BY INLCPU01 ASC
```

*Figure 6-13  SQL for viewing the Add Times test results*

Note that the input is Stats Flat data from file (G_STATSFPG). Stats Flat was used instead of Stats Hierarchical because the test programs were all run from one job and determining who called whom was not an objective. Also, there is collection overhead that affects the result. That overhead is lower for Flat data than it is for Hierarchical, so there is less bias on the results.
Figure 6-14  Add Times Test ILE Program’s RPG source

```
DVAR              s              9B 0 inz
C                   DO        1000000
C                   ADD       1             VAR
C                   ENDDO
C                   RETURN
```

Figure 6-15  Add Times Test OPM Program’s RPG source

```
C                     Z-ADD0         VAR    150
C                   DO        1000000
C                   ADD       1             VAR
C                   ENDDO
C                   RETRN
```
In Figure 6-16, each program’s name indicates the program type (ILE or OPM), the data type and length, and the optimization level as *NONE (no indication) or *FULL (the name ends with _O).

The data values are the Inline CPU time used by each program. They are adjusted as shown in Figure 6-17 on page 158 to show only the Add instruction’s time in the Add Time column.

This view shows each program’s Inline CPU time in microseconds. Two tests were run to provide some measure of consistency.

Note the first and fourth rows from the top and the last and fourth from the bottom rows. These are the base programs that provide the non-Add (or infrastructure) costs in the corresponding test programs.

The calculated add times are shown in the table in Figure 6-19 on page 160.
Figure 6-17 shows the Add instruction times as the result of each program’s Inline CPU time minus the corresponding base program’s Inline CPU time.

The integer Add operation was the fastest. The optimized program was approximately twice as fast as the non-optimized integer programs. Following that, the Packed Decimal operations have a wide range of times that depend primarily on their operand type, length, and optimization usage.
Figure 6-18 provides a graphic look at the range and patterns of the RPG Add instruction time variation as the data type, length, and program optimization settings change. The graph is packed too tightly for detailed information. All of the input data is in Figure 6-19 on page 160.

In some cases, it may improve performance to convert an data base file input field from Packed to Integer, work with it, and convert it back to Packed when the program is through with it. Part of the gain will depend on how often the variable is used. If it is low use, it probably is not worth converting it.

On the other hand, if you are coding in C++, which has the ability to Cast variables to different data types, it may be worthwhile. You should run this type of test on a proposed program and data variable configuration to see what runs the fastest, especially in a compute intensive function.
The values have been changed by subtracting the non-Add instruction overhead to provide the best information available using PEX Stats Flat collection about the relative Add instruction times for different data types and lengths.

**Important:** The times shown are inflated by PEX Stats collection. A true performance test can be done only by using a measurement procedure that does not introduce the bias of additional cost due to collection overhead.

**Disclaimer:** These results were obtained on a system running an uncontrolled environment. You may or may not get similar results in your run time setup.

When changes are made to the System i hardware, the i5/OS, or the application programs, it is a good practice to have a set of repeatable performance test cases to establish a set of performance differences and expectations between the current and previous run time environment. The hardware and operating system can change and affect run times. Application changes often induce unexpected performance changes. It is a good idea to be prepared in case the upgraded system's performance has noticeable changes.

<table>
<thead>
<tr>
<th>Program name</th>
<th>Add Time</th>
<th>Program name</th>
<th>Add Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILEINT20_O</td>
<td>622</td>
<td>ILEPKD13</td>
<td>22,316</td>
</tr>
<tr>
<td>ILEINT20</td>
<td>751</td>
<td>OPMPKD13_O</td>
<td>22,545</td>
</tr>
<tr>
<td>ILEINT10_O</td>
<td>827</td>
<td>ILEPKD15_O</td>
<td>28,953</td>
</tr>
<tr>
<td>ILEINT10</td>
<td>2,564</td>
<td>OPMPKD15_O</td>
<td>30,603</td>
</tr>
<tr>
<td>OPMPKD14</td>
<td>15,876</td>
<td>OPMPKD15</td>
<td>31,744</td>
</tr>
<tr>
<td>OPMPKD8_O</td>
<td>16,196</td>
<td>ILEPKD15</td>
<td>32,444</td>
</tr>
<tr>
<td>OPMPKD9_O</td>
<td>16,453</td>
<td>ILEPKD17_O</td>
<td>32,721</td>
</tr>
<tr>
<td>OPMPKD5_O</td>
<td>16,461</td>
<td>ILEPKD19_O</td>
<td>32,795</td>
</tr>
<tr>
<td>ILEPKD13_O</td>
<td>16,481</td>
<td>ILEPKD18_O</td>
<td>33,646</td>
</tr>
<tr>
<td>ILEPKD7_O</td>
<td>17,402</td>
<td>OPMPKD17_O</td>
<td>34,728</td>
</tr>
<tr>
<td>ILEPKD14_O</td>
<td>17,501</td>
<td>OPMPKD19_O</td>
<td>35,114</td>
</tr>
<tr>
<td>ILEPKD6_O</td>
<td>17,508</td>
<td>OPMPKD17</td>
<td>35,476</td>
</tr>
<tr>
<td>ILEPKD8_O</td>
<td>17,522</td>
<td>OPMPKD19</td>
<td>35,711</td>
</tr>
<tr>
<td>ILEPKD5_O</td>
<td>17,544</td>
<td>OPMPKD18_O</td>
<td>36,662</td>
</tr>
<tr>
<td>ILEPKD9_O</td>
<td>17,590</td>
<td>OPMPKD18</td>
<td>38,047</td>
</tr>
<tr>
<td>OPMPKD7</td>
<td>17,736</td>
<td>ILEPKD18</td>
<td>45,782</td>
</tr>
<tr>
<td>ILEPKD5</td>
<td>17,831</td>
<td>ILEPKD19</td>
<td>46,258</td>
</tr>
<tr>
<td>ILEPKD8</td>
<td>17,848</td>
<td>ILEPKD17</td>
<td>46,317</td>
</tr>
<tr>
<td>ILEPKD9</td>
<td>17,921</td>
<td>ILEBIN4_O</td>
<td>60,523</td>
</tr>
<tr>
<td>OPMPKD7_O</td>
<td>18,382</td>
<td>ILEBIN4</td>
<td>63,574</td>
</tr>
<tr>
<td>ILEPKD14</td>
<td>18,683</td>
<td>ILEBIN5_O</td>
<td>65,709</td>
</tr>
<tr>
<td>ILEPKD6</td>
<td>18,686</td>
<td>ILEBIN5</td>
<td>68,823</td>
</tr>
<tr>
<td>OPMPKD13</td>
<td>18,686</td>
<td>ILEBIN6_O</td>
<td>70,491</td>
</tr>
<tr>
<td>OPMPKD14_O</td>
<td>19,231</td>
<td>ILEBIN6</td>
<td>73,751</td>
</tr>
<tr>
<td>OPMPKD6_O</td>
<td>19,391</td>
<td>ILEBIN9_O</td>
<td>79,824</td>
</tr>
<tr>
<td>OPMPKD9</td>
<td>19,414</td>
<td>ILEBIN8_O</td>
<td>79,852</td>
</tr>
<tr>
<td>OPMPKD6</td>
<td>19,542</td>
<td>ILEBIN7_O</td>
<td>79,921</td>
</tr>
<tr>
<td>OPMPKD8</td>
<td>19,602</td>
<td>ILEBIN9</td>
<td>81,896</td>
</tr>
<tr>
<td>OPMPKD5</td>
<td>19,806</td>
<td>ILEBIN7</td>
<td>81,953</td>
</tr>
<tr>
<td>ILEPKD7</td>
<td>21,008</td>
<td>ILEBIN8</td>
<td>81,955</td>
</tr>
</tbody>
</table>

*Figure 6-19  Add times results for all test cases*
This chapter discusses System i performance topics that you may be able to apply to your PEX Stats analysis work.
7.1 Overview

The i5 has different performance characteristics than some of its predecessors. In some areas, the key factors that influence system performance have changed. Some applications and types of work are more complex and the need for security, faster data access, and continuous availability impose new demands on the system.

At one time, the effort to achieve proper system performance was encumbered by inadequate processing power and insufficient main storage capacity. Today's systems do not lack for CPU capacity, main storage capacity, disk read and write cache, or total disk storage size. As a result, some application's performance can be improved by hardware changes. Other applications require detailed analysis to determine what needs to be modified so the application can take advantage of more powerful hardware. That is where knowing how to use PEX Statistics, Job Watcher, or PEX Trace and Profile is necessary.

7.2 General concerns

One of the basic questions is “what do you look for in the performance data that need answers to achieve optimal performance”? This may include:

- What resources do the jobs need to process a request for work?
- What amount of each resource do the jobs need?
- At what rate are the resources consumed?
- How is contention for resources managed and is it working optimally to meet the application’s objectives?

One step is to know what resources jobs need to process their work. There are many different types of resources:

1. Disk storage space for data base files and other objects.
2. Main storage to completely contain the programs and data, for example, their “working set” so they can avoid high page fault rates.
3. Rapid access to the disk devices so necessary data can get in and out of main storage without undue delay.
4. Access to data objects, such as data base files that may be being used by multiple concurrent jobs and sometimes exclusively by only one job.
5. Timely access to adequate amounts of processor time so the data can be processed.
6. Access to the outside world (the Web, workstations, ATMs, kiosks, POS terminals, and so on) to accept and respond to requests for work.

Performance analysis requires an understanding of the system’s management methodology for different types of resources and how it is working for your application.

1. Managing access to different resources requires different methods.
   a. Some resource requests must provide the full amount needed before the job can continue, such as access to a data file.
   b. Other resource requests can sometimes allow sharing, such as disk I/O operation processing.
   c. Some requests can be served with an interleaved approach that guarantees an amount to all requests, such as processor time or disk space.
2. When a resource request does not provide the needed amount over a period of time, it causes the job/thread to take more time than it should to complete a work request. This has a cumulative downside effect on satisfying the system’s service level agreements; to avoid these situations, the causes need to be evaluated to see if alternative resolutions might apply, for example, how much delay is tolerable?

7.3 Possible candidates for improvement

One critical performance item is the disk I/O request rate per disk arm. When the disk I/O request rate reaches a certain level, the resulting disk I/O queuing effect disrupts the ability to have stable and consistent flow of work requests. When that happens, the user experiences erratic service time swings that result in reduced throughput, responsiveness, and customer satisfaction.

7.3.1 Disk I/O requests

The amount of data transferred for many system disk I/O requests, both reads and writes, is a single disk sector (4096 bytes). Many performance studies show that multiple sector reads and writes are a much smaller percentage than the single sector I/O. Some applications are built (and some old ones have been changed) to do “blocked” disk I/O, that is, they do multiple sector transfers to or from disk. Using blocked instead of unblocked disk I/O results in more efficient system operation characterized by higher throughput rates and lower overall resource costs. It also enhances the application’s ability to scale its workload growth with hardware growth with minimal transition costs to the customer and application development.

Many System i functions optimize their output requests by blocking the data, as much as they can, up to 128 KB. This happens a lot in Save/Restore and Journal processing. Similarly, a number of system functions use input blocking when possible.

The system provides user interfaces to specify preferred processing options and parameters for data base processing and data base related functions, such as journaling and System Managed Access Path Protection (SMAPP). Proper use of the file create and override command’s blocksize parameters and the use of sequential processing where possible are zero-cost enablers of higher throughput rates.

Optimal performance depends on having data in main storage when it is needed. It is important to understand and apply the proper disk parameters, both in disk I/O request processing and in disk space management in ASM.

The length of a disk I/O request queue is often a limiting factor to throughput. The queue length is dependent on a number of factors:

1. The number of requests issued per unit time. In queuing equations, this is called the arrival rate.
2. Number of concurrent jobs requesting disk I/O operations.
3. System functions that issue multiple overlapping concurrent disk I/O read or write requests which, in some cases, can overload the disk arms and I/O caches and inhibit performance.
4. Object contention that inhibits jobs from proceeding and issuing I/O requests.
5. The amount of data to be transferred in or out by an I/O request. This can often be controlled by user/application settings that are beyond this book’s scope.
6. Whether or not the data is in contiguous disk sectors, thereby enabling hardware level look ahead for input data transfer. As systems get older, they often have a significant amount of data fragmentation. Over time, the amount of contiguous disk sectors available for file allocation or extension is less and multiple non-contiguous sectors are needed to satisfy space allocation requests.

7. For write requests, the size of the I/O subsystems write cache memory.

8. Not having enough disk controllers on an IOP.

9. The use of RAID 5 or mirroring.

Discussion

Some disk I/O operations are productive, while others are non-productive. Non-productive requests include re-reading data that has been recently read but, because it has not been used for a period of time, has been overlaid and is no longer in main storage. This may happen due to high page fault rates on an overcommitted system.

If a job's disk I/O operation rate goes up because of increased non-database page faults, it is possible that giving the job's pool more main storage would help. However, there are a number of times when assigning more main storage to a pool does not cause its fault rate to go down.

One rule of thumb for changing the amount of a resource is “if more does not help, take away ten percent of what you last increased it by and quit adding more. If more helps, keep adding it until the previous rule applies”.

Study your system's disk I/O request rate compared to system throughput. If the number of disk operation's per widget (or whatever your product is) processed is going up, that means the disk I/O costs are increasing. Discover why and where it is happening and determine what needs to be done to stop the increase. If the rate tends to increase consistently over time, at some point the rate will be so high that the I/O subsystem will not be able to handle the demand.

You need to be able to predict the workload rate at which this peak may occur so you can proactively prevent potential performance problems. Use PEX Stats to find the programs that have high disk I/O request counts and concentrate on finding the reason(s) and try to reduce their disk I/O rate.

Sometimes a file may have a very high ratio of deleted to active records.

Note: A target guideline value of 0.1 for the (Number of Deleted Records divided by the Number of Active Records) is a reasonable target, especially for files that have duplicate keys or are the target of a truncated key search.

As an example, some applications have log files to record recent activity and the old records (perhaps older than some number of days) are read and deleted every day. One user had a log file with millions of records in it, but only a few thousand were active and the rest were deleted. Every day the deleted record count increased and day by day the applications ran longer and longer. Tracking the costs, which in this case was the number of disk I/O operations by the processing jobs compared to the number of file related transactions, provided an early warning of creeping processing slowdown.

In one case, a user pre-loaded a file into a private pool using the Set Object Access (SETOBJACC) command. This practice was quickly abandoned because the time to pre-load it was greater than the processing time saved. The real answer was to get rid of the file's deleted records, either by recreating a new version of the file or by reorganizing it.
7.3.2 Disk space usage

Application design and implementation affect performance depending on how different types of processing needs are handled by the application and the system. When there are hundreds or thousands of jobs/threads running concurrently on a system, there can sometimes be very high peak load demand for system services, such as disk I/O, CPU, object contention resolution, and space management.

1. Create/Delete DB files or members or DB file indexes.

Many applications have high rates of creating, using, and then destroying “scratch files” in their processing. Creating and deleting space does not seem like much overhead, but when you consider how many concurrent jobs can run in today’s powerful processors, gigantic main storage capacity, and large numbers of disk arms, many of them are frequently allocating and deallocating disk space. High create, extend, and delete request rates cause high contention rates for single points of reference, such as a disk drive’s available free space information. When one job is changing this information, the effect is often that other jobs have to queue for that information. This in turn affects a job’s ability to meet its processing time objectives, which translates into stretching its response time as a result of slowing the job’s database record processing rate.

Very high request rates for disk space allocation and deallocation are one of the top causes of degraded performance.

2. Full DB Open/Close is one of the most frequently encountered causes of performance problems, even though it has been the topic of many learned IBM and vendor presentations, discussions, and papers over the years. One of the main reasons that high Open/Close rates cause performance problems is a secondary effect; they are one of the major causes of high rates of space allocation and deallocation requests.

The PEX Stats data for one recent customer shows that, among other things, the operating system database file open and close programs (QDBOPEN and QDBCLOSE) were being called (506625 + 461752) 968377 times in 574 seconds. The result shows that the rate of disk space allocation and deallocation requests was 1687 times per second. Assuming there are 30 disk arms on the system, this results in about 56 requests per second per arm.

This number is just for the database file open’s and close’s space management requests, each of which creates or destroys an object. Each request requires a synchronous disk read and a synchronous disk write. At the same time, there are other disk I/O operation requests that need to use the disk arms while other jobs are going through space allocation so the disks are keeping busy.

3. Disk I/O Caching: A major characteristic of disk I/O cache is the performance advantage gained when:

   a. The requested input data is in the cache and a disk read is not needed.

   b. When a write request goes to the cache and is signalled complete. It is sent to the disk later.

As an aside, why not just install a bigger CPU? Is that an answer to the problem? That approach assumes that an increase in processor power eliminates the problem by making everything run faster. The short answer is that the processor isn’t the limiting factor; the speed of the disk data transfer is what is important. There have been significant improvements in disk access time technology, one of which is write cache. Given adequate write cache storage, most synchronous disk write requests are signaled as complete within one or two milliseconds or less. What the request complete signal means is that the data is now in the write cache and will be written to disk sometime later. This
allows the requesting job to proceed without waiting for the data to be written to the disk drive.

Adding more processor often results in higher disk I/O request rates because a job finishes processing the input data faster and issues its next disk I/O request sooner. This drives the disk utilization (which is a function of the request rate) up and the result is that eventually the speed of the disk access becomes the primary limiting factor to throughput. At high rates, even a one millisecond disk I/O request response time can cause disk I/O queuing problems.

From the calculation above, an average rate of create/destroy requests of 1687 per second isn’t always average. Studies have shown that these types of requests, as well as many others, have a normal distribution pattern. Statistically that means that about 10 percent of the time the request rate is approximately triple the average rate and about 30% of the time the request rate is about double the average rate. That's one of the reasons that performance “spikes” occur. One performance improvement objective is to reduce the disk I/O rate as much as possible. There are a number of ways to do that which have been the subject of many IBM and 3rd party papers, classes and Common presentations for some time.

At very high rates of disk contention the processor usage can encounter a hard limit because most of the active jobs are waiting for disk I/O. Some systems have experienced this limit at about 50 to 60 percent CPU utilization and can't be driven any higher regardless of the number of jobs in the system. The throughput and processing rate flattens out regardless of the CPU power. Adding more main storage doesn't help a job that requires a lot of disk accesses; one solution is to ensure that you have enough write cache because it often reduces the performance impact. The best solution is to keep the demand (e.g. the request rate) below the knee of the queuing curve.

A few suggestions for reducing disk I/O demand:

- Change programs are often called and go through full termination and require re-initialization each time they are called. This can eliminate high Open/Close rates, which has a secondary effect of reducing the number of ASM requests, which in turn reduces disk I/O request rates.
- Use *CALLER instead of *NEW in your ILE programs if they are called more than once by the same caller.
- For files with a lot of Add activity, use a separate file ODP in the program. Open it for Output only and use sequential processing, and record blocking. This enables the next point about SMP.
- Use SMP (Symmetric Multi Processing) to enable parallel index maintenance during keyed physical file blocked Adds file index maintenance processing.
- Do not fill an output subfile with months or years worth of data. Many inquiries are satisfied with the last ten day’s of data.
- Use Journal Blocking for data base processing.
- Use remote journaling to avoid having to “harvest” the journals by the High Availability software. The journal records are sent to the backup system prior to its being written on the target system.

4. Estimate Data Space Index Key Range (ESTDSIKR): When you see this instruction running, it is probably not causing any problem and in fact can be helping. It is often run as part of SQL Query Optimization to estimate the time to run a query. The instruction does not need any Auxiliary Storage Management (ASM) functions, but the result of running it may.
If the Query Optimizer's evaluation, used by SQL, OPNQRYF or Query400, indicates that a data base file index is needed to optimally run the query, a data base index is built. Some index builds take a lot of time (sometimes minutes) to create and allocate the disk space and much more time to populate it. In many cases, a high number of jobs have had to wait a long time (sometimes minutes) for a job's SQL or other query index create to finish before they could apply changes to the underlying physical file.

PEX Stats shows whether or not a data base file index build occurred (the MI CRTDSINX instruction). If there are a high number of index creates and high in-line elapsed time, investigate further using the Database Monitor, SQE Plan Cache, Index Advisor, or other tool, to identify the columns and table used to create the index, prior to moving on to using Job Watcher. Creating a single index (or modifying and recreating an existing index) may eliminate multiple temporary index creates.

Use Job Watcher to see if other job's waits are caused by a job that is building a data base index. During a data base file index build, any job that wants to change the file has to wait until the build finishes.

5. Extend DB files: When a data base file "fills up", as a result of added record processing, that exhausts the file's available free space, more free space has to be added to the file before processing can proceed. In other words, the file capacity has to be extended. (Note that in this case either there are no deleted records that can be used for new records or the program is doing sequential blocked output add processing which does not reuse deleted records.)

When a file is full, the i5/OS data base add program (unblocked QDBPUT or blocked sequential QDBPUTM) invokes ASM to make the file bigger. Within a specific application in which there are many copies of the same job running, other jobs can encounter heavy contention when the concurrent processing jobs are doing periodic file maintenance, such as daily wrap-up processing. While each job is busily adding records to the file, their processing can encounter a temporary abrupt halt while ASM finds, allocates, and records the additional space for a highly shared file. The delay, although brief, occurs frequently enough to affect throughput and impose non-optimal overall performance.

Eliminate the file extend wait condition by pre-allocating the file space when it is created or when adding a new member to the file. Use the data base Create File (CRTPF) or Add Member (ADDPFM) command's ALLOCATE parameter to specify that you want to reserve (for example, allocate) all the space the file is going to need (until it is created again). Your customer may balk at this saying that it costs too much in disk space; the reality is that the space will eventually be used as the file grows, but in the meantime, the applications are not going to encounter a file extend wait condition. When file space is pre-allocated, there are no file extend processing waits and the application is a big step closer to an optimally performing and scalable set of jobs.

One characteristic of frequent file extension waits is that no matter how many concurrent jobs are started, the throughput will flatten out and cannot be driven any higher. The effect of this is that the throughput rate also flattens and in some cases it drops slightly. The throughput will rise linearly up to a point, perhaps around 60% CPU use, but then flattens regardless of the number of concurrent jobs added. More CPU does not help; the cause is a serialized disk I/O wait condition.

This is similar to the high disk I/O rate problem discussed earlier, but it is characterized by reduced average disk I/O rates and reduced CPU usage as many jobs encounter an internal wait on the disk that is providing the new space for file extension.
6. Compare the use of i5/OS data base program’s QDBPUT and QDBPUTM. QDBPUT does unblocked, single record Adds to a database file. This is the assigned default that data base programs must use when a data base file is open for I/O (both input and output).

QDBPUTM can do sequential blocked adds to a database file. The current maximum output blocksize is 128 KB. One way to improve performance in a program that has a lot of added records is to change the program to open the file twice, for example, have two File Descriptions; one for input only, the other for output only.

The output file should have an override (OVRDBF) that specifies Sequential Only *YES and blocking. Set the blocksize to the number of records needed to fill a 128 KB block. This approach provides optimum index maintenance processing (if SMP is used) for the added records and optimum disk performance by performing blocked writes.

7. FEOD: The Force End of Data operation does two operations: it takes a new record from the program’s file buffer and puts it into the data base and synchronously writes the record to disk. This requires the job to wait until the disk write finishes. In PEX Stats, one indication of this is *ENSOBJ MI instruction usage. To avoid the synchronous write wait, change the RPG program’s FEOD to FEOD(N). This approach still adds the record to the data base file so other jobs can access it, but does not force the physical record to disk with a synchronous write.

8. ENSOBJ: Ensure Object MI Instruction, which synchronously writes data to disk. There are multiple paths to using the MI Ensure Object instruction. They can be the result of:
   - Use of data base file’s non-zero FRCRATI/O value, which forces the file’s changed records out of main storage. It should not be used, as there’s no benefit to FRCRATI/O greater than zero on today’s systems; using it slows down the job and provides no added value functionally.
   - The last job closing a data base file that is open for update forces all of the file’s changed records that are in main storage to disk storage. The writes must complete before the close operation is complete and control returns to the program.
   - The Set Object Access command’s *PURGE option removes a file from main storage. Any changed pages in main storage are purged to disk.
   - Delete File: To improve the performance of an application that is doing a lot of file deletes, one alternative might be to replace the use of create and delete operations by having a permanent scratch workfile and use the Clear Physical File Member command when it is no longer needed (or ready to use it again).

This reduces system overhead if these are scratch files and are always discarded and if the system is not at security level 40 or above (which imposes additional processing overhead on a Clear Physical File member).

9. Creating/Deleting Spool Files and Job Logs: These objects require disk space, but the performance of file create/delete is not a significant concern. This is because the operating system provides a number of empty spool buckets ready to be used when a job needs to create a print file. The system aggressively stays ahead of the users in pre-creating spool files.

10. Program Activation and Deactivation: There is some associated ASM overhead with this activity, but it is not as intrusive as other, secondary processing that happens as a result. High use indicates that there may be related processing, such as higher than normal data base file Full Open/Close processing.
11. Indiscriminate use of ILE *NEW Activation Group: When program A calls Program B, the called program needs a work area. Building that is part of the program activation process. In many applications, A makes many calls to B and therefore B's implementation and run time environment are critical to performance. There are three ways to provide the activation working areas for B:

a. Use the calling programs existing space (create option *CALLER).

b. Create a separate, explicitly named work area for B the first time it is called (create option *NAMED).

c. Create a separate work area for B each time it is called and destroy it each time B returns (create option *NEW).

The best one to use is (a); there is usually no need to allocate a new object if you can share the calling program's space.

The second option (b) is almost as good as the (a) option, except that it requires a small amount of additional time the first time B is called to create a named object that will be reused on subsequent calls to the program.

The last one (c) should *not* be used in some cases. It can lead to significant performance impact if the program is called a lot, does not run very long each time, and fully terminates each time it is called. It is not recommended, as it requires invoking ASM twice each time the program is called to allocate or deallocate the new object and its disk space.

12. Lot of jobs with very short total run time:

This is similar to using *NEW for program activation. The difference is that it is a whole job, including the programs, that needs disk space. There are transaction driven applications that create a job to process the data and then terminate the job, sometimes hundreds of times a second. The job management overhead plus the cost of the application program's opening and closing files each time is expensive. The Job Watcher data may show that the system is often pushed to the limit. High rates of job initiation and termination can put a much higher disk I/O request rate on the system and can lead to more slowdowns.

One approach is to structure processing jobs to be message driven through Data Queues or user defined queues. It is much faster and efficient to wait on a data queue message instead of closing all the files and terminate the job and then have to restart a new job a few milliseconds later.

Another possibility is to use pre-started jobs. See the System i Work Management document at the following web location for more information:


7.3.3 Main Storage

Having enough main storage is always important. Main storage's purpose is to provide a buffer or cache between the disk and the CPU. Each of these hardware components has some amount of high speed storage; their discriminating factors are their relative size, speed, and cost. The CPU cache is extremely fast and expensive, and therefore relatively small compared to main storage and disk I/O caching.

Main storage contains the data that the CPU needs next or contains the data that the CPU no longer needs and can be discarded (or kept around for subsequent use and perhaps written to disk if it is changed). It is shared between jobs when possible, it is moved between pools on a system when needed, and it is given to other partitions at times.
Job A running in pool X can access main storage in pool Y, but most of A’s disk I/O read requests bring the data into job A’s pool X. One exception is object loading using the SETOBJACC command, which can be directed to load an object into any main storage pool. Main storage is owned by the storage pool it is contained in, not by the jobs running in the pool. There are commands to explicitly move some amount of main storage from Pool X to Pool Y.

A job’s main storage usage is not explicitly instrumented and main storage automatic allocation to pool X or Y (for example, tuning) is based on inferential management using pool page fault rates. Sometimes the Storage Management High Priority Page Out task (SLIC task SMPO001) runs to sweep changed or “old and aged” pages out of main storage when the amount of available main storage in a pool goes below a threshold.

The only main storage information in PEX Stats is the Page Fault Start event, a counter that can be optionally collected. Note that because it is collected and reported by program within a job, it provides explicit information about what job(s) and program(s) caused the page fault.

More detailed information as to what object and pool a page fault is related to is available in Job Watcher, PEX Disk I/O, or Task Switch traces.

### 7.3.4 Processor

The only performance tool that explicitly provides an estimate of program CPU use is PEX Stats and it is not accurate enough for prediction or capacity planning.

PEX Stats collects and reports CPU use by both program and job. Job level CPU use is available in other tools, such as Job Watcher, Collection Services, PEX Task Switch trace, and Print Transaction Report. PEX Profile provides relative CPU use between instructions, PEX and TPROF provides it between programs.

Do not attempt to correlate the CPU measurements between these tools. Their recording methodology is such that an accurate comparison usually is not possible, and there is no way to know if you have achieved accuracy.

A program’s CPU time shown by PEX Stats includes data collection overhead. This cost is indeterminate, so there is not necessarily a high percentage of accuracy when trying to correlate the CPU time in the same program in the same test case from multiple collections.

Having stated that, it is reasonable to compare the CPU times between runs to see what relative differences there are. It is not recommended to use the Stats CPU time value in a performance prediction or capacity planning effort.

### 7.3.5 DB2 SMP

The SMP parallel degree setting controls the level of parallelism for both CQE and SQE.

Input processing with CQE uses DBL3 tasks, while SQE uses threads (off the user’s job). In general, it takes a lot less threads than tasks to provide the same level of throughput. For example, a CQE full table scan might use 255 tasks, where the same table scan with SQE will use 16 threads (on an 8 way system).
The QQUERYDEGREE parameters mean:

- *NONE* disallows any parallelism (with some exceptions, such as scanning an EVI or building column statistics, both of which will use some parallelism by default).
- *OPTIMIZE* allows CPU parallelism and each optimizer will determine the appropriate degree based on number of CPUs and fair share of main storage in the job's pool.
- *MAX* allows CPU parallelism and the optimizer will determine a maximum degree based on number of CPUs and all of main storage in the job's pool. For CQE, this is generally the lesser of total number of disk units or 255. For SQE, this is generally 2x CPUs.

For CQE, I/O parallelism must be enabled by degree using the parameters *IO*, *OPTIMIZE*, or *MAX*. For SQE, I/O parallelism is enabled by default.

More information about DB2® SMP is available at:

http://ibm.com/servers/enable/site/education/abstracts/4aea_abs.html
Appendix A. Building a PEX Definition for PEX Stats collection

This section discusses how to create a PEX Definition using the iDoctor GUI. First, bring up PEX Analyzer window, Select PEX Analyzer, and select Create PEX Definition from the submenu (see Figure A-1).

Create PEX Definition puts you in total control of what gets specified. Using Create PEX Collection lets you choose between the Basic or Advanced path.

The primary differences are:
1. The Basic path default is PEX Stats. After that, you choose Flat or Hierarchical implicitly.
2. The Basic path allows job selection.
3. The Advanced path uses a pre-built PEX Definition and does not provide job selection.
4. You must use Create PEX Definition if you want to specify additional event counter assignment. See “PEX Event Counts Collection specification” on page 178 and “Collection Properties: Event Counters Definition” on page 189 for more information.
Create PEX Definition Welcome menu

Figure A-2 shows the Create PEX Definition Welcome menu.

![PEX Definition Wizard - Welcome - Rchassq1](image)

Figure A-2  iDoctor PEX Definition Wizard Welcome menu

This is an information only window; click Next to proceed.

Create PEX Definition Wizard - Type Selection

Figure A-3 shows the Create PEX Definition Wizard - Type Selection window.

![PEX Definition Wizard - Type Selection - Rchassq1](image)

Figure A-3  iDoctor PEX Analyzer Create PEX Definition Wizard - Type Selection

This window’s default selection is Statistical data. Make sure Statistical is selected and click Next.
Create PEX Definition Statistical Options

Figure A-4 shows the Create PEX Definition Statistical Options.

![Figure A-4](image.png)

Complete the following fields:

- **Definition name**: Give it a unique name or you will replace an existing definition with the same name.
- **Description**: Optional, but good to remind you what the collection contains.
- **Data Organization**: Select either the **Flat** or **Hierarchical** option button.
- **Thread options**: Leave it at the default.
- **List all threads/tasks**: If selected, information about all threads is collected for the Collection Properties.
- **Select Specific events for counting**: Check this if you want to specific additional event counting on a subsequent option window. Available only on Create PEX Definition.

**Note**: If you want PEX Event counts collected along with the Stats data, then you must explicitly create the collection's PEX Definition before starting a collection. This function is not available on the Basic path.
Create PEX Definition Program Bracketing Events

Figure A-5 shows the Create PEX Definition Program Bracketing Events window.

Use the default values shown by this menu, use *PRC instead of *MENTRYEXIT, or what combinations are needed by IBM Service. Then select **Next** to proceed to the next menu.

If *PRC is used instead of *MENTRYEXIT, then for ILE programs, the Stats Hierarchical Analysis output has a Program Entry Point row immediately followed by a Procedure entry row.

**Note:** If the **Select specific events for event counting** box was checked in the Options menu, the next menu in the sequence is the Events menu. If not, the next menu is the Job Selection menu.

**MI Program Events**

There are three mutually exclusive options in the MI Program Events options box shown in Figure A-5:

1. **MENTRYEXIT:** If this default value is selected, programs that were created with the ENBPFRCOL(*ENTRYEXIT) parameter value can have data collected for them. This applies to ILE service programs, because their default creation setting is ENBPFRCOL(*PEP). The *PEP setting has no effect on service programs, because they do not have a Program Entry Point (PEP). To collect data over service programs, two things must happen:
   a. The service program must be created using the parameter setting ENBPFRCOL(*ENTRYEXIT)
   b. The MI Program Bracketing Event selection MENTRYEXIT or PRC must be used.
2. PRC: If PRC is used as a MI Program Bracketing Event, an event is recorded at service program procedure entry and exit.

3. None: Nothing is collected.

**PEX Event Counts Collection specification**

The system’s PEX Data Collection (PDC) can collect information about many different types of events. Event data includes:

1. Event Type: What type of event?
2. Event Counts: How many times did it happen?
3. Event specific data, such as disk unit, job ID, and so on.
4. Event related data, such as issuing job and time of day.

You can specify that the first two, Event type and Event counts, can be collected and reported during a PEX Stats collection, either Flat or Hierarchical. The data is recorded for each program or MI Complex instruction that it occurs in. The specific event type and counter number is specified in the Stats PEX Definition.

Events occur (are caused by) a program or MI instruction. When they happen and if event counts are enabled, the program/instruction’s related event counter is incremented. The event counter values are part of the PEX Stats default report descriptions.

Use these counts to see which of them may be causing additional overhead for one reason or another. The counts are recorded as both Inline and Cumulative values for each of the four counters.

More information about PEX Events is in “PEX event definitions” on page 190.
Figure A-6 shows the Event Selection specification menu.

To define what event data is to be collected during a PEX Stats collection, do the following in the Event Select window in Figure A-6:

1. Select the event Category (application, basic, disk I/O, storage, and so on). Click the list to open it to view the different categories.
2. Select the Category event(s) you want to include in a single counter; iterate through these steps to go to other categories and add other events to a specific counter.
3. Select the Counter to accumulate the selected events counts into. There are four counters (1 - 4).
4. Click Add Event(s) to assign the events to the counter. The counter-category-events appear in the Events to Collect list.
5. Iterate through these steps to define the different events and event counter assignments. Click Next when you are through.
Event Collection Definition example: Page Faults

Figure A-7 shows the Event Selection Specification Wizard example setting.

Do the following:
1. Select Category Fault events.
2. Select Event Page Fault Start.
3. Select counter 3 to receive the counts for this event. Multiple events can be counted into the same counter number. The individual event contributions are not preserved.
4. Select Add Events to add the event to the Events to collect list.
5. Note that the counter-category-event-Short name values are filled in.

It is unnecessary to count both Page Fault Starts and Page Fault End events. You are probably interested in the number of page faults that occurred within a program.

Clicking Next takes you to the job selection initial menu Job Task Options.

A suggested Event Counter specification

The following events are one possible configuration you could use. They cover a number of areas of interest to a performance analyst.

The counter number assignments in the list are arbitrary; you can use any of the four counters. The suggested events are:

Counter 1
- Storage Events: Create and Extend Segment.
  This shows object create and extend activity (ASM).

Counter 2
- Storage Event: Delete Segment.
  This shows object delete activity (ASM).

Counter 3
  Shows which programs/MI Instructions are getting the most faults.
Counter 4

Base Events: Activation Group Activate Program and Activation Group Create.

If there is a lot of MI *DEACTPG activity in a PEX STATS collection, go to the Stats Flat job view to find the jobs and programs in which they are occurring.

The counter 4 events show if there is a lot of program activation and deactivation activity. As a result, you could check to see if there is a lot of Deactivate Program activity and whether or not any of the called (not the current line’s program) programs use *NEW for the activation instead of *CALLER or *NAMED.

Create PEX Definition Job Task Options

Figure A-8 shows the Create PEX Definition Job Task Options.

![Create PEX Definition Job Task Options](image)

*Figure A-8  iDoctor PEX Analyzer Create PEX Definition Wizard - Job/Task selection*

See the discussion in 2.7.6, “Job/Task selection” on page 36 for detailed information about job selection specification.
PEX Definition summary

Figure A-9 shows the PEX Definition summary.

This is the last menu in the Create PEX Definition process. It summarizes the information needed for performing data collection.

Note that the last line, beginning with QSYS/ADDPEXDFN, is the command string for generating a PEX Definition from a green screen or a CL program with the Add PEX Definition command.

In summary, this PEX Definition specifies a Stats Flat collection over all QPADEV* jobs with user name JOE_USER and collects PEX Events for ASM, Program Activation Management, and page faults. Selecting Finish saves the PEX Definition in file QUSRSYS/QAPEXDFN.
iDoctor and PEX Analyzer options

This appendix discuss some specific options and capabilities of iDoctor and PEX Analyzer.
Library Submenu Options

The PEX Analyzer Library submenu in Figure B-1 assists in managing the contents of a PEX library.

![Figure B-1 Library submenu](image)

From this menu you can:

- **Explore**  
  Show a view of all the PEX Collections in the library.

- **Create PEX Collection**  

- **Delete all PEX Analyses**  
  Remove the collection members from all PEX analysis files in the library. The files are not deleted. The files are named G_.... . There may be other, PEX analysis files (but not Stats files) in the library depending on what you have used it for.

- **Delete all PEX Collections and Analyses**  
  Remove the collection and analysis members from all PEX collection and analysis files in the library. The files are not deleted. The collection files are named QAYPE..., and the analysis files are named G_.... . This will affect PEX Stats, PEX Profile, and PEX Trace files.

- **Copy**  
  Copy the library's contents to another library.

- **Save**  
  Save this library as a *SAVF in another library.

- **Transfer to**  
  Send the library to another system.

- **Clear**  
  Remove all objects from the library and keep the library.

- **Delete**  
  Delete the library and all its content.

- **Rename**  
  Give the library a different name.

- **Properties**  
  Show information about the library.

Note that the Copy, Delete, and Transfer options are also provided on the collection member view. That view can be seen by clicking on the library submenu Explore option.
If you want to send a collection to another system, either use a collection submenu or, alternatively, copy the collection to a new (or empty) library, save that new library, and send it to whatever destination you want.

Another method is to FTP it to a PC, zip the file to make it even smaller, and perhaps convert it to a self extracting file, and e-mail it to another destination. The destination can unzip it, create a new library and *SAVF on the system, and then FTP it to the system and restore the library.

To use the Transfer option, the “to” system has to be in iDoctor's My Connections list or you need the “to” system's IP address. Select the options from the Transfer menu prior to running the transfer.

**Collection member options**

A collection member submenu has similar options to Copy, Delete, or Transfer a PEX data collection member. Use this to work with individual members.

**Work with PEX Definitions options menu**

If you delete PEX collections, there may be some other housekeeping needed. If the deleted collections were built with the Basic path, the collection's PEX definitions (with the same name as the collection) are still in the system. You can work with them (that is, delete or perform other actions) from the PEX Analyzer menu.

To view the PEX Definitions that PEX or iDoctor know about:

1. Go to the iDoctor PEX Analyzer view (Figure B-2).
2. Right-click **PEX Analyzer**.
3. Select **Work with PEX Definitions**.

![Figure B-2  PEX Analyzer Work with PEX Definitions](image-url)
After selecting **Work with PEX Definitions**, the view in Figure B-3 lists all of the PEX Definitions in file QUSRYS/QAPEXDFN on the system you are connected to.

You can reorder any column by clicking its description (header). Clicking produces an ascending sort, right-clicking produces a descending sort.

When you right-click a PEX Definition, an options menu appears (Figure B-4).

From this menu, you can:

- **Create PEX Definition**
  Use the selected definition as the base for creating a new one.

- **Change PEX Definition**
  Modify the selected definition.

- **Create PEX Collection**
  Use the selected definition to set up a collection.

- **Delete**
  Remove the selected PEX Definitions (You can select one or more rows).

- **Properties**
  Show the PEX Definitions creation values.
Appendix B. iDoctor and PEX Analyzer options

Collection properties

A PEX Stats Collection has related data that can be viewed as needed. To view it, right-click in a collection's data view and select Properties → Collection (Figure B-5).

Figure B-5   View Collection Properties from a data view

Getting to the collection properties

There are three collection properties available;

**Query**

Shows the current views Query name, its SQL SELECT statement, and its data source (library, file and member name).

**Analysis**

Varies depending on the view. For Stats data, it shows the name of the job that the data was collected for.

**Collection**

There are a number of collection properties. Select from the Tab strip (in Figure B-6 on page 188) to select what you want to see.
Collection properties summary

Select the Collection Jobs tab to see information about the job, such as job name, main storage pool it ran in, Elapsed run time, CPU time, and so on.

Job Properties: What did Job X do

Selecting the Collection Jobs tab in Collection Properties window shows the window in Figure B-7 with the jobs run time information.
Collection Properties: Event Counters Definition

Use this tab to see the collection’s event counter specifications. More information about Event Counters is in “PEX Event Counts Collection specification” on page 178.

![Collection Properties Events Tab](image1)

Selecting the Collection Properties Events tab brings up the view shown in Figure B-9.

![Stats PEX event counter assignments](image2)

This view shows what counter contains counts of what events. The Stats event counters record the number of times each of these occur for each program that data is collected for in both the Flat and Hierarchical collection modes.

For example:

**Counter 1**
Contains the count of the number of Create Segment (Subtype *CRTSEG) and Extend Segment (Subtype *EXDSEG) MI Instructions that occurred for the total collection.

You are viewing Collection Properties, so the values are the total counts for all job in the collection.

**Counter 2**
Contains the count of the number of Delete Segment (Subtype *DLTSEG) and Truncate Segment (Subtype *TRUNCSEG) MI Instructions that occurred for the total collection.

**Counter 3**
Contains the number of page faults that occurred (Started -- Subtype STR). Counter 4 contains a count of the number of disk I/O read and disk I/O write events during the collection.

**Counter 4**
The number of disk I/O read and write starts. Note that page fault starts are included in this count. Page faults occur higher up the logical chain of events, so a disk I/O start event is at the bottom of that chain.

The terms in the Subtype column, such as *CRTSEG, can be found as shown in “PEX event definitions” on page 190.
PEX event definitions

This section shows you how to find and see the definition of PEX events on the IBM System i performance information Web site.

Go to http://www.ibm.com/servers/eserver/iseries/perfmgmt/traceevents.html and you will get a Web page showing the PEX events. There are a number of event categories. For example, one of the interesting event categories is stgevt, the storage event.

On the IBM Web site window, click stgevt in the line that looks like:

stgevt(*crtseg, *fndsegsiz, etc.)

A table appears that shows the individual Storage Event Descriptions. Counting these events shows whether or not there are a lot of MI Object Space Management requests (for example, one event per request) and how often the objects grow or shrink and how much space is acquired or relinquished each time. Counting the event does not show all that information, but a high count tells you that more analysis with PEX Trace of the Storage Events will probably be worthwhile.

There are other PEX event types that are often of interest. The IBM Web site page shows other event categories of interest. The rest of this section discusses some of these category’s events.

Storage Event descriptions

The following Storage Events are the ones that we are interested in:

**CRTSEG**  
*Create Segment. An event is generated whenever a segment is created.

**DLTSEG**  
*Destroy Segment. An event is generated whenever a segment is destroyed.

**EXDSEG**  
*Extend Segment. An event is generated whenever a segment is extended.

**TRUNCSEG**  
*Truncate Segment. An event is generated whenever a segment is truncated.

**Note:** Most of the TRUNCSEG events have the length specified as zero. These entries are not of interest in Stats or other user related performance analysis. All PEX ASM Trace Analysis excludes the zero length events. There is not a lot of application level use of TRUNCSEG and you cannot affect systems use of this function.

**Suggestion:** If you assign Storage Events to the PEX STATS Definitions event counters, assign CRTSEG and EXDSEG into one counter and DLTSEG event into another one if you are interested in the number of times there was an acquisition or release of object space. Usually, most activity is divided between object creation or extension and you should assign object deletion to its own counter.

Storage events are performed on behalf of all jobs and tasks on the system by the SLIC Auxiliary Storage Management function (ASM). High ASM request rates usually impact system performance, sometimes noticeably. A lot of performance analysis effort is directed towards explicitly reducing the number of creates and deletes or implicitly reduce the number of calls to ASM.
Some of the areas where this can be accomplished are:

- **Explicit effect:** Reduce the number of file opens and closes. Each open and each close create two objects.

  When a system is getting two hundred to two thousand (or more) Full Open/Close operations per SECOND, it affects performance.

- **Implicit effect:** Pre-allocate a data base file’s disk space for files that will have a lot of added records activity and not a lot of record deletion. Another option (which is a far second place) is to make the secondary allocation amount much larger.

There are a number of other storage events but, because of the significant amount of collection cost involved, *do not use them* except in extremely rare cases (and after you have experienced them in a non-customer-production environment). They have extremely high counts and do not apply to ASM usage. Use them only with the direction and guidance of a knowledgeable IBM Service Representative.

Additional discussion about space management is in 7.3.2, “Disk space usage” on page 165.

### Job Event (JOBEVT) descriptions

**LWSTR**  
Job Long Wait Start. An event is generated whenever a start of a long wait occurs. It also means that the job has given up its Activity Level.

Note that a “Short Wait” could have occurred and the Long Wait is a result of the Short Wait Time-out, usually after two seconds. If there are a lot of them, it with contribute to a higher Inline Elapsed Time value for the program.

**INELIGIBLE**  
Job Ineligible. An event is generated whenever a process moves from an active state to ineligible. Frequent occurrences may indicate system overload or high priority jobs bumping lower priority jobs out of the activity level. It is unusual to see a lot of this in a “batch” job.

**ACTIVE**  
Job Active. An event is generated whenever a process moves from ineligible or wait to active. If there are no ineligible counts, this value might indicate the start of processing a transaction in a non-batch environment. It is unusual to see a lot of this in a “batch” job.

**MPLPOOLCHG**  
Job MPL Pool Change. An event is generated whenever the job is changed to run in a different main storage pool. Note that this is not the result of the QTSEPOOL system value setting; that is a separate event TOBCHMPLPOOL. You are not expected to see many of these.

**TOBCHMPLPOOL**  
Job To Batch MPL Pool. An event is generated whenever a MI process is being implicitly moved to the batch MPL pool defined for it. Occurs only if System Value QTSEPOOL is enabled. Not expected to see many of these.

**TSLEND**  
Job Timeslice End. An event is generated whenever a Job’s External time slice end occurs. If the job has been in a long running MI instruction, the job may have greatly overrun its time slice setting. High counts are unexpected.
Time Slice End can result in job priority demotion by the system’s Dynamic Priority Scheduling (DPS) algorithms. In a busy system, it may be a basic cause of some jobs not providing “normal” transaction response time. If so, the solution may be to run the job in a higher priority range rather than increasing its external time slice.

**MPLLEAVE**

An event is generated whenever the job looses its Activity Level. This can happen at the start of a Long Wait, External Time Slice End, or by being pre-empted by a higher priority job running in the same main storage pool.

If it happens a lot, you can try increasing the pools activity level by 10%. If that does not work, do not keep increasing it, but look for other causes.

Pay particular attention to the page fault counts in other jobs that are running in the same pool.

### BASE event descriptions

**ACTGRPACTPGM**

Activation Group Activate Program. An event is generated whenever an ILE program activation begins.

**ACTGRPCRT**

Activation Group Create. An event is generated whenever an ILE program activation group is created.

**ACTGRPDLT**

Activation Group Delete. An event is generated whenever an ILE program activation group is destroyed.

High counts for these three preceding events are an indication of improper program implementation. Within a job, a program should be activated once. If it is happening multiple times, the program setup should be investigated. If the counts are relatively high, it impacts performance. Expect the count to be either one (for a well built program) or equal (or close) to the Times Called count for an improperly built program.

**EXCP**

Exception. An event generated when an exception occurs below the MI interface.

**MIEXCP**

MI Exception. An event generated when an MI exception occurs or when an EXCP event is being resignaled above the MI Interface.

If there are high counts for these two events, additional analysis can be done. Check the Joblog to see what MI Exceptions were logged. In addition, examine the program they are occurring in, probably in a separate debug mode test or with the Trace Job commands STRTRC/ENDTRC. This may help to understand what exceptions are occurring and to determine how to reduce their frequency or eliminate them.

### Disk (DSKEVT) event descriptions

**READSTR**

Read Start. An event is generated whenever a start of a physical read occurs.

**WRTSTR**

Write Start. An event is generated whenever a start of a physical write occurs.
Page fault (FAULT EVT) event descriptions

**STR**
Start of Fault. An event is generated whenever a page fault occurs.

This event is used to discover where the cause of high paging environments is. This (or PEX Disk I/O and Program Call trace) are the only tools and events that show what programs cause the page faults. Job Watcher data shows what jobs and objects are page faulting.

PEX Disk I/O can show what object and disk unit the fault occurred on; Combining PEX Disk I/O and MI Entry/Exit trace can provide program, object, and disk arm information.

Lock (LCK EVT) event descriptions

**LWEND**
Seize Lock Long Seize Wait End. An event is generated whenever long seize wait completes.

Remember these can be very short and there might be lots of them. This does not necessarily indicate a problem. You can investigate them in better detail with Job Watcher, which often can provide conclusive results. If not, the next step is to use PEX Task Switch Trace for a one to two minute period to record their occurrence and discover what job/thread/task is holding the object.

Operating system (OSEVT) event descriptions

**DBIO**
Database I/O. An event is generated whenever a database logical data I/O is done by a job/thread. These events are analogous to the Collection Services data base logical I/O count fields JBLWT, JBLRD, and JBWRT in the QAPMJOB file. High values are not always an indication of high disk I/O counts.

**DBOPEN**
Database Open. An event is generated whenever a database full file open or close is done by a job/thread. This should match (or be close to) the QDBOPEN programs Times Called count value.

**DTAARA**
Data Area. An event is generated whenever a data area access (read or write) is done by a job/thread. This value should follow the Times Called count for program QWCCCHVC.

**DTAQ**
Data Queue. An event is generated whenever a data queue access is done by a job/thread (message send or receive). This value should follow the Times Called count for programs QSNDDTAQ and QRCVDTAQ.

Java (JVAEVT) event descriptions

**OBJCRT**
Java object create. An event is generated whenever a Java object instance is created.

**GBGCOL**
Java garbage collection. An event is generated whenever Java is doing garbage collection.

**GBGCOLSWEEP**
Java garbage collection sweep. An event is generated whenever the garbage collection sweep completes.

**THDCRT**
Java thread create. An event is generated whenever a Java thread is created.

**THDDLT**
Java thread delete. An event is generated whenever a Java thread is destructed.
THDSSP  Java thread suspend. An event is generated whenever a Java thread or thread group suspend is started.

THDRSM  Java thread resume. An event is generated whenever a Java thread or thread group suspend is resumed.

THDWAIT  Java thread wait. An event is generated whenever Object:wait() used for thread synchronization is started.

THDNFY  Java thread notify. An event is generated whenever Object:notify() used for thread synchronization is started.

THDNFYALL  Java thread notify all. An event is generated whenever Object:notifyAll() used for thread synchronization is started.

THDSTTCHG  Java thread state change. An event is generated whenever a method that causes a thread state transition (for example, a thread block) is started.

CLSLOAD  Java class load. An event is generated whenever a Java class is loaded.

CLSUNLOAD  Java class unload. An event is generated whenever a Java class is unloaded.

LIBOPR  Java class library operation. An event is generated whenever a Java native method library is loaded or unloaded.

TFMSTR  Java transform start. An event is generated whenever a Java program object generation step is started.

LCKSTR  Java lock. An event is generated whenever a synchronize lock is requested.

UNLCK  Java unlock. An event is generated whenever a synchronize lock is released.

JVAEND  Java end. An event is generated whenever a previous Java event is completed.

Journal (JRNEVT) event descriptions

STRCOMMIT  Start of journal commit cycle. An event is generated when a journal commit cycle starts. During a commit cycle, the associated data base file changes are written asynchronously to the journal. At the end of the commit cycle, they are written synchronously.

ENDCOMMIT  End of a journal commit cycle. Generated when the journal commit cycle ends, usually at the direction of an application program.

STRROLLBACK  Start of rollback. An event is generated when a journal rollback starts. System performance could be impacted if there are a lot of rollback operations. This operation interferes with “normal” data base changes journaling.

Rather than counting Journal events with PEX Stats Event Counters, look at the collection's MI Commit and Decommit instructions activity. A MI Commit is the same as the ENDCOMMIT journal event; a MI Decommit is the same as a STRROLLBACK event.
Exception (EXPCCHEVT) event descriptions

**TUNE**
A record of the storage management expert cache information and shows how storage management monitors I/O and memory usage. An event generated by the Storage Management component of Expert Cache.

**TUNEDB**
Contains the file classifications of the data base files and the logical and physical read data used to make the classifications. An event generated by the database component of Expert Cache.

The fastest path in and out of collection: STRPEX and ENDPEX

After you have created a PEX Definition, you can use the STRPEX command to start a collection session and ENDPEX to end the collection. You can do this either from a command line or within a CL program.

It is much easier to create a PEX Definition from iDoctor's GUI than it is to use the ADDPEXDFN command. Create the definition first from iDoctor's PEX Analyzer GUI and then use these commands.

You could use these commands to start collection, run a test scenario from your workstation job (QPADEV...), and then end the collection. In this case, the PEX Definition's Job Selection can be “current job”. The command descriptions are on the IBM.COM Web site. Click the links below to get to them.

The **STRPEX** command discussion is at:

The **ENDPEX** command discussion is at:

After ENDPEX finishes, use iDoctor's PEX Analyzer to run a PEX Stats Analysis over the collection.

iDoctor performance tips

This discussion shows you how to improve sluggish iDoctor performance.

If iDoctor requests are not running quickly enough because the system you are running on is experiencing performance problems, you may be able to improve iDoctor performance and your productivity by changing the run priority and external time slice of the iDoctor server job QZDASOINIT.

To do this:
1. In the PEX Analyzer’s main window, right-click the left hand windows **PEX Analyzer** line.
2. On the menu, right-click **Properties**.
3. On the iDoctor Properties menu, select the **iDoctor Client Jobs** tab.
4. As shown in the window in Figure B-10:
   a. Change the server jobs Run Priority value to a lower number. Make it lower than the application's high priority jobs (that usually run at priority 19 or 20). This allows the iDoctor server jobs to get the CPU immediately when they need it.
   b. Change the server jobs CPU time slice value to 10000 (ten seconds). Because much of the iDoctor work uses SQL over the PEX data, the queries could take longer than the jobs default external time slice for the QUSER job class.
   c. Select OK to exit the properties window.
5. On the iDoctor main menu, select File → Exit.
6. Respond OK to the iDoctor message, as shown in Figure B-11.
7. Select the iDoctor File → Exit or Close the iDoctor window to end iDoctor, as shown in Figure B-12 on page 197.
8. Restart iDoctor. The iDoctor's server job priority and time slice value are set to the new values.

![Figure B-10](image1.png)
*Figure B-10  Changing and applying iDoctor server job priority and timeslice*

![Figure B-11](image2.png)
*Figure B-11  iDoctor shutdown/restart message*
Appendix B. iDoctor and PEX Analyzer options

Why do this

Implementing these recommendations allows you to more quickly do your job to get the customer back to good performance. If your activity is inhibited because of resource contention on the customers system, you are not working as fast as you could. One of the ways to do this is to get the CPU when you need it (that is, the priority change) and to keep the CPU once you get it (that is, changing your external timeslice so you do not get kicked out of the activity level before your current query finishes).

You are not using a lot of the customer's CPU time, but when you need it, you need it now and not in five or ten minutes. Besides, they are paying a lot for performance analysis services and you should work as efficiently as possible.

Finding a program module’s ENBPFRCOL setting

Use the Display Service Program command to find a service programs ENBPFRCOL attribute.

For example, to find the attribute of the QSYS library program QP0APFS, run the command:

DSPSRVPGM SRVPGM(QSYS/QP0APFS)
From the result screen, press Enter to go from Display 1 of 10 (Figure B-13) to Display 3 of 10 (Figure B-14) (as designated in the screens upper right hand corner)

**Display Service Program Information**

**Display 1 of 10**

<table>
<thead>
<tr>
<th>Service program</th>
<th>QP0APFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>QSYS</td>
</tr>
<tr>
<td>Owner</td>
<td>QSYS</td>
</tr>
<tr>
<td>Service program attribute</td>
<td>CPPLE</td>
</tr>
<tr>
<td>Detail</td>
<td>*BASIC</td>
</tr>
</tbody>
</table>

**Service program creation information:**

- Service program creation date/time: 05/13/05 13:40:07
- Export source file: S000009800
- Export member: S000009800
- Activation group attribute: *CALLER
- Shared activation group: *NO
- Current export signature: D8D7F0C1D7C6E24040404040

Press Enter to continue.

**Display Service Program Information**

**Display 3 of 10**

<table>
<thead>
<tr>
<th>Opt</th>
<th>Module</th>
<th>Library</th>
<th>Attribute</th>
<th>Creation Date</th>
<th>Optimization Level</th>
<th>Debug Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>QPOAVFS</td>
<td>QBUILDSS1</td>
<td>CPPLE</td>
<td>11/30/03</td>
<td>*FULL</td>
<td>*NO</td>
</tr>
<tr>
<td>QPOAVNOP</td>
<td>QBUILDSS1</td>
<td>CPPLE</td>
<td>11/30/03</td>
<td>*FULL</td>
<td>*NO</td>
<td></td>
</tr>
<tr>
<td>QPOAVNOD</td>
<td>QBUILDSS1</td>
<td>CPPLE</td>
<td>11/30/03</td>
<td>*FULL</td>
<td>*NO</td>
<td></td>
</tr>
<tr>
<td>QPOAVNET</td>
<td>QBUILDSS1</td>
<td>CPPLE</td>
<td>11/30/03</td>
<td>*FULL</td>
<td>*NO</td>
<td></td>
</tr>
<tr>
<td>QPOAVS1</td>
<td>QBUILDSS1</td>
<td>CPPLE</td>
<td>05/13/05</td>
<td>*FULL</td>
<td>*NO</td>
<td></td>
</tr>
<tr>
<td>QPOADMON</td>
<td>QBUILDSS1</td>
<td>CPPLE</td>
<td>11/30/03</td>
<td>*FULL</td>
<td>*NO</td>
<td></td>
</tr>
<tr>
<td>QPOAUTIL</td>
<td>QBUILDSS1</td>
<td>CPPLE</td>
<td>11/30/03</td>
<td>*FULL</td>
<td>*NO</td>
<td></td>
</tr>
<tr>
<td>QPOASNA</td>
<td>QBUILDSS1</td>
<td>CPPLE</td>
<td>11/30/03</td>
<td>*FULL</td>
<td>*NO</td>
<td></td>
</tr>
</tbody>
</table>

Press Enter to continue.

F3=Exit  F11=Display character signature  F12=Cancel

**Figure B-13** i5OS Display Service Program command: step 1

**Figure B-14** i5OS Display Service Program command: scrolling down
Enter a 5 (Display description) in the Opt column row(s) of the Module(s) you are interested in. In this example, it is only QP0AVFS (Figure B-15).

Press Enter.

Display Service Program Information

Service program ................. : QP0APFS
Library ....................... : QSYS

Optimization level .............. : *FULL
Maximum optimization level ...... : *FULL
Debug data ..................... : *NO
Profiling data ................. : *NOCOL
Number of procedures .......... : 10
Number of procedures block reordered .. : 0
Number of procedures block order measured .. : 0
Enable performance collection ....... : *PEP
Teraspace storage enabled .......... : *YES
Storage model .................. : *INHERIT
Module created on .............. : V5R3M0
Module created for ............ : V5R3M0
Object control level .......... : 33341244

Press Enter to continue.

F3=Exit  F12=Cancel

The line you are interested in is:
Enable performance collection ....... : *PEP

This shows that the program was created with the default ENBPFRCOL value and therefore is not properly enabled for collection. Because it is a service program, the ENBPFRCOL value used at creation time must be *ENTRYEXIT, not *PEP.

It needs to be changed with the following command, assuming you have the authority to do it:
CHGSRPVM SRVPGM(libname/srvpgmname) ENBPFRCOL(*ENTRYEXIT *ALLPRC)

Including *ALLPRC changes all procedures in the service program.
Querying and graphing PEX Statistics data

This appendix shows you how to define and run user defined queries and to graph PEX Stats data. It includes how to use the two iDoctor build SQL statements interfaces to query PEX Stats Analysis database files (files whose names start with G_STASxxx). One interface is for the non-SQL expert user, the other is for the knowledgeable SQL user.
Querying PEX Stats data using iDoctor’s GUI

Two different SQL usage methods are provided with iDoctor that enable you to create and run SQL-based queries over PEX Stats Analysis output file data. They are:

- Query definition interface (for novice SQL users)
- SQL Query view (for expert SQL users)

iDoctor’s Query definition interface

Tables and graphs can be created within the PEX Stats component using the iDoctor Query Definition menus. Use these menus to define what data columns are to be retrieved, from what collections, and how the results are displayed (as either a table or graph). Using the Query Definition interface over an existing view’s SQL statements enables a novice SQL user to customize the query for the active table or graph within an iDoctor Data View. All iDoctor Stats data table and graph views have a Query Definition Menu interface available to let you work with the SQL for that particular view.

The iDoctor Query Definition interface lets you build a SQL statement to meet your needs based on your file, member, field, and record selection criteria.

**Note:** File, member, field, and record are iDoctor’s terminology. The SQL terminology is table, data set or user created Alias, column, and row, respectively.

You can examine the resulting SQL statement from a iDoctor Data Viewer properties window to see how it was built; you can apply any subsequent changes with Query Definition. The Query Definition interface restricts you to one file when building a user defined query. This restriction does not apply when using iDoctor's New SQL Query interface discussed in the section “iDoctor's New SQL Query view interface” on page 217. From there you can use Join, sub Select, and other SQL Select options.

**Note:** The Data Viewer is part of the iDoctor for System i tool. You can use it to display tables and graphs over any described data base file on your system. You can have as many Data Viewer windows open at one time as you want.

When you open a view, there is an option to use an existing Data Viewer (by viewer number) or to open a new one. The viewer number is in the upper left corner of the window.

Accessing the Query Definition interface

To get to the Query Definition interface, right-click in a data view and select the **Query Definition** menu. See Figure C-1 on page 203.

Member selection

To get to the menu where you can change a query’s input data member:

1. Right-click in the Data View.
2. In the resulting menu (Figure C-1 on page 203), select **Query Definition**.
3. Select **Member Selection**.
Appendix C. Querying and graphing PEX Statistics data

This displays the Query Definition menu with the Member Selection tab selected. The Query Definition menu has multiple tabs for defining different parts of the query. In this example, choose the Member Selection tab shown in Figure C-2. Member Selection is used to view or modify the views Stats data collection file or member.

On this menu, select the Change Member button to see the collection’s member name list.

4. To change the query input member, select a list member and click OK. See Figure C-3.
Field selection and order

Select the **Field Selection** tab (Figure C-4). The fields are listed in the same order they occur in the file. Use this menu to define which fields are shown and to specify their order of appearance.

![Query Definition: Field Selection tab menu view](image)

To change the field order:

1. Highlight one or more adjacent or nonadjacent field rows
2. Select and drag them to the desired location within the list.
   
   The field is inserted immediately following the field you drag them to. It also works for multiple nonadjacent fields selection.

A field appears in the data view if its **Show?** column box is checked. Unchecked fields are not shown.

For example, to put the MI Complex Instruction field right after the Procedure Name:

1. Select the field.
2. Drag it towards the target field until the target field (for example, Procedure Name) is highlighted.
3. Release the cursor and the new field order is shown (Figure C-5).

As mentioned above, this also works to move multiple selected fields.

Figure C-5   Moving fields in the Field Selection menu

One way to remove a large number of field names from the view is to highlight the group and toggle their Show? box. To do this to adjacent fields:

1. Select the first one you want to remove.
2. Hold down the Shift key.
3. Drag the cursor to the last field name in the group. When you release the cursor, all of the fields will be highlighted.
4. Click **Toggle Selected**.
This toggles the Show check box setting (between checked and non-checked) for each field in the group of selected field names, as shown in Figure C-6. This can be very handy when you want to hide or show a large number of fields at once.

![Query Definition](image)

**Figure C-6  Toggle selected fields**

**Note:** This procedure to hide a number of fields is useful when you are defining a Group By query and there are very few output fields from the original data. Most of the fields are new fields built with SELECT xxxx “AS” yyyy statements.

### Creating new fields

The Field Selection menu is also used to create new fields (Figure C-7 on page 207):

1. Select the row that you want the new field to follow.
2. Click the **New Field** tab to add a new row to the list of fields. To add $n$ rows, click **New Field** $n$ times.
3. Select the row to be edited (for example, external name, internal name, or SQL specification).
4. Select the **Edit Field** tab to go to the Edit Field's menu (Figure C-8 on page 207).
On the Edit Field menu, specify:

1. Field Description: A description for the new field that also is used as the output column header value.

2. Field name: The internal SQL field name to use.

3. SQL expression: Enter the field’s SQL expression.

This example creates a new field called ELPCALL with a description of Elapsed Time per Call. The new field ELPCALL is the result of the SQL expression that divides the Inline Elapsed Time field’s value by the Times Called field’s value. If Times Called (the denominator) is zero, the result is indeterminate.

If you do not define it, SQL will define a default length and number of decimal digits for the result field. This example explicitly defined a five digit field with two decimal digits.

4. Select **Apply** to save this field definition.
5. If there are more fields (for example, rows) to be defined, move the cursor to their row in the Field Selection menu, select that row, and select the **Edit** tab to go back to the Edit Field menu.

6. When you are finished with the definitions, exit the menu using the Windows Exit icon (the X in the menu's upper right hand corner).

After exiting the Edit Field menu, the results appear in the Field List (Figure C-9).

Select **OK** to exit the Query Definition and run the query or select another Query Definition tab to do more definition.

![Figure C-9 The Field List after applying the new field definition](image-url)
Figure C-10 shows the new field result.

![Figure C-10](image)

The new field shows the average elapsed time per call of the program/MI instruction. It can be used in record selection to select or omit different values or value ranges. It can also be used in Sort to sort the view, perhaps from highest to lowest value.

**Selecting specific input data**

Click the **Record Selection** tab to define which records to view.

For our example, we want to look at only non-i5/OS programs. This criteria is defined to mean that you want:

1. Programs that do not reside in a library whose name starts with Q.
2. The MI Complex Instruction field is blank, that is, the row represents a program and not a MI instruction.

**Define the first selection**

1. As shown in Figure C-11 on page 210, select the Field name list Library Name (LIBNAM) entry.

   This list contains the name of all of the fields.

2. From the Operator pull-down menu, set the operator to **Does not start with**.

   The list entries are context sensitive; only valid operators are shown for a field.

3. Set the Value to ‘Q’ (the single quotes must enclose each character data variable; multiple entries are separated by a single blank).
4. Click **Add Filter** to add the criteria to the Record Selection Filter List. Note the default Boolean condition is ‘AND’. You can set it depending on your selection criteria.

**Define the second selection**

1. Select the **Field name list MI Complex Instruction (MICPXNM)** entry.
2. From the Operator pull-down menu, set the operator to **Starts with**.
3. Set the Value to a single blank ‘ ‘.
4. Click **Add Filter** to add the criteria to the Record Selection Filter List (Figure C-11).

Figure C-11 shows the Query Definition Record Selection definition results.

To select only the non-QSYS library programs, you must explicitly exclude the MI Complex Instructions. This is because their row’s LIBNAM field also does not start with Q.

**Specifying sort keys**

**Note:** *Always* specify sort order (that is, specify an Order By clause). Why? SQL does not guarantee any particular order of the output data if you do not sort it.

SQL implementation may merge sets of selected data and if there is no Order By, the view sequence could be different between runs.

On the Query Definition menu, click the **Sort By** tab to define a view’s sort sequence.

In this example, to sort the report by the new field ELPCALL in descending order:

1. Set the Field parameter to ELPCALL(ELPCALL).
2. Set Sort Order to Descending.
3. Click **Add Field** to add the criteria to the Field Sort Sequence List, as shown in Figure C-12 on page 211.
4. Repeat this sequence for additional sort keys.
Appendix C. Querying and graphing PEX Statistics data

Figure C-12  Query Definition: Sort By tab

This is the end of this example’s definition; select OK to run the query.

The result is in the view in Figure C-10 on page 209.

Group By

This section starts with the intermediate representation of the final view where the Group By results are shown. The example shows how many times the program QDBxxxxxx was called by each job represented by its job number.

The view in Figure C-13 uses the Stats Flat summary by job file in which there are three fields selected:
1. The job number
2. The program name
3. The number of times the program was called

Figure C-13  Starting view for Group By discussion
The SQL selects *all* i5/OS data base programs whose names start with QDB. This includes QDBOPEN (full db file open) and QDBGETKY (data base read by key).

The query results show how many times each program was called within each job. These results infer that two Group By clauses are used.

*Before* applying the Group By selection, the SQL looks like Example C-1.

**Example: C-1  SQL to select all Jobs, i5/OS Data Base (QDB...) programs and their Call count**

```sql
SELECT QTSJNB, PGMNAM, QSTINV FROM dbname/G_STATSFJB WHERE PGMNAM LIKE 'QDB%' ORDER BY INLCPU DESC
```

From a view generated by the SQL in Example C-1, go to Query Definition and select the Group By tab to define which records will be visible in the report (Figure C-14).

Group-by queries are valid only when the fields on the field selection page comply with the SQL rules for running Group By within a SQL statement.

Any fields that are not part of the group-by clause must be summarized so they exist in the field selection, or the query will not run.

**Figure C-14  Query Definition: Group By**

**Restriction:** The Query Definition interface is built by parsing the contents of an existing SQL statement. This parsing works well for many queries, but it does not acknowledge all types of SQL syntax. It will parse most SQL select statements containing JOINs, but there are some complex statements that cannot be parsed (such as UNIONs). Although a query can be parsed that contains JOINs, the type of JOINs and the files being JOINed cannot be changed using iDoctor's Query Definition interface.

To do this through the iDoctor GUI, use the New SQL Query icon and view described in “Accessing the New SQL Query view” on page 218 and enter the SQL statement as needed.

The Query Definition menus can be used to adjust the WHERE, ORDER BY, and GROUP BY clauses of the *outermost* part of the SQL statement. However, any ORDER BY, WHERE, or GROUP BY clauses for *subqueries* in the SQL statement are not configurable through this interface.
Figure C-15 shows the Group By fields added to the query definition.

![Group By fields added to the query definition](image)

The use of Group By requires that other changes be made to the SQL to accommodate the new logic. These changes are:

1. Add a result field to contain the total call count
2. Change the sort to Order By the following two keys
   a. Job number. In a real world example, you would have job name and user ID also. They are omitted here for confidentiality reasons.
   b. Total Call count (probably descending)

The results of this user-defined query are shown in Figure C-16. The first column shows the job numbers for each batch job that was running during the collection. Subsequent queries can inspect these jobs in more detail.

![Query Group By definition results](image)
This result helps your data analysis by identifying jobs where additional data and analysis is needed. What is apparent is:

1. The job (number 628579) performed 2118 full data base opens and closes. The collection properties show that the total collection time and the job run time was 575 seconds.

   This many file Open/Close operations strongly suggests that the job’s programs are fully terminating and subsequently re-opening their data base files each time they are called. This can be checked by looking for *DEACTPG or *DESAGP MI instructions. For more information about this topic, see 3.3.6, “Markers: MI Complex Instructions analysis” on page 72.

2. The job is doing keyed reads (QDBGETKY), though not on every Open/Close combination.

3. The job is occasionally changing a record (QDBUDR -- update).

4. The job is occasionally adding a record to the file (QDBPUT -- unblocked write add).

   Updates and adds require data base file index maintenance. You know it is a keyed file because of the keyed reads and therefore has at least one index to maintain. Updates and single record adds are the most expensive in terms of index maintenance costs. Much more efficient processing is available by doing blocked adds and using symmetric multi-processing (SMP).

Figure C-17 shows the SQL for the view in Figure C-16 on page 213.

```
SELECT QTSJNB, PGMNAM, SUM(QSTINV) AS TIMESCALLED FROM /G_STATSFJB WHERE PGMNAM LIKE 'QDB%' GROUP BY PGMNAM, QTSJNB ORDER BY QTSJNB ASC, TIMESCALLED DESC
```

*Figure C-17  SQL for the view in Figure C-16 on page 213*
Saving a new query definition

Save your query definition to use later by right-clicking in the view and choosing **Query Definition → Save As** (Figure C-18). All non-system scoped Query Definitions are saved on the user's Windows system (the PC client). Multi System query definitions are stored on the server.

![Figure C-18 Saving your new query definition](image)

To save the SQL in the upper portion of the window as an iDoctor User Defined Query:

1. Move the cursor to the lower portion of the view.
2. Right-click in the view.
3. Select **Query Definition**.
4. Select **Save As**.
5. Follow the save prompts.

Figure C-19 shows an example of the Save Query Definition Interface. You can assign a description for your query definition up to 250 characters long.

![Figure C-19 Save Query Definition](image)
The Save Query parameters are:

**Component**
Specify PEX Analyzer here. If you were in another iDoctor tool, you would select that tool’s component name (for example, Job Watcher).

**Version**
Usually the current VRM.

**Scope**
Sets the query’s range of availability. It can be for all connected systems, the current system, the current library, or only the current collection.

**Description**
User defined. Make it meaningful and easy to find.

**Replace**
Check the *Replace existing query definition* box to replace the saved query definition with the current one. This check box is visible only if the table view was created from a user-defined query definition.

If replace is not checked, there can be multiple UDQs with the same name. To see which is yours, you can run it or look at its Date Time value to see when it was built.

All non-system scoped UDQs are stored on the iDoctor user’s PC client in a local database. To determine the location of this database, use the context menu of the PEX Analyzer icon, and select *User-defined reports → Select local database*.

UDQs scoped to All Systems or Current System can be used against any library on the system that contains the files that the query uses.

**Working with query definitions**
After a query definition (a UDQ) has been created, it can be displayed and opened from a collection’s user-defined queries folder, as shown in Figure C-20.

![Figure C-20 Work with User-defined queries](image-url)
From here, you can open (that is, run the query), delete it, set its scope, or view the query properties, which includes its SQL statement, as shown in Figure C-21.

**Figure C-21 User-Defined Query properties**

On this menu, you can select and copy the SQL statement and paste it into a new location, such as an i5/OS STRSQL command view, a document or e-mail, or an iDoctor New SQL menu and then modify and run it. This is a timesaving approach if you have a large file to query but you want to modify it before you run it.

If you use it in a STRSQL view, you will need to set up the proper file OVRDBF command(s) parameters.

**iDoctor’s New SQL Query view interface**

The SQL Query view, which is for more experienced SQL users, lets you dynamically execute and display the results of a SQL SELECT query. The view enables you to work with more than one file using JOINs or UNIONs; you must know how to write SQL, as there is no iDoctor assistance with building the statements.

**Note:** When using this interface to change/create SQL, please be aware that you need to have good knowledge of SQL query and that the iDoctor interface supports SELECT statements only. It does not support all SQL, such as CREATE TABLE.

This iDoctor interface is essential if you wish to build up your own queries that JOIN multiple tables together.

The queries created with this view may be saved and restored for later use. Their definitions can be viewed and manipulated, in part, with iDoctor’s Query Definition interface. A benefit of using iDoctor’s **Query Definition** with the SQL Query view is that any changes to the query definition (which changes the SQL statement) are visible immediately in the top portion of this window.
Accessing the New SQL Query view

The SQL Query view can be accessed within a Data Viewer by selecting the New SQL Query (left-most) icon from a Data Viewer toolbar, as indicated by the arrow in Figure C-22.

![Data Viewer toolbar: New SQL query view icon](image)

When you click this icon, the System selection window appears (Figure C-23). Select OK if the name represents the system over which you want to run your SQL statements. Otherwise, change to the appropriate system and then click OK.

![System selection window](image)

In Figure C-24, the top portion of the view is the area where you enter SQL statements. The bottom portion is where the SQL execution results output appears.

To execute your query after entering or editing it, right-click the top portion of the view and choose Execute or press F4 (Figure C-24). You can re-execute your query at any time.

![Executing your SQL statements](image)

From the top section of this view, you can:
- Modify and execute the query. You can also run it by pressing F4.
- Any selected item can be cut or copied from it to the clipboard.
- You can paste into the window.
Fit the SQL statement into the viewable part of the window by selecting **Word wrap**, which toggles the word wrap option.

Use **Make Lowercase** or **Make Uppercase** to set the case for any selected data.

When you start to run the query, and there is more than one collection in your library, the member names window in Figure C-25 appears. Choose the member to process by selecting it and clicking **OK**.

This menu appears once for each FROM statement in your SQL before proceeding to execution.

![Choose member for REDBOOK1/QAPYJ2W1DE](image)

*Figure C-25  Choose the database file member to be processed by the SQL*

When the query is finished, its results appear in the bottom portion of the table view. From this view, the typical copy-to-clipboard, query definition, graph definition, and find functions that are standard in a table view are available.

**Graph Definition interface**

With iDoctor, you can define graphs over a data view or to modify an existing graph view. Do this by creating an iDoctor graph definition, which contains the control information used to build a user-defined or iDoctor component-supplied graph. Graph definitions are stored similarly to the way query definitions are stored.

A graph definition defines what is needed to display the graph, including a reference to the query definition needed to produce the graph's data. When a graph view is saved, both the current query definition and the current graph definition are saved.

The interface is accessible by either:

- Defining a new graph definition by right-clicking in the view and selecting **Graph Definition → Define New**, as shown in Figure C-26 on page 220.
- Using the Graph Definition pop-up menu in a user-defined graph view.
User-defined graphs

As the name states, user-defined graphs (UDG) are created by you, the user, and saved into a graph definition. Its settings are called its graph definition. The SQL statement (query) behind the table view is also used as the query behind the graph. Such a graph is created from a table view using the Graph Definition → Define New context menu, as shown in Figure C-26.

![Figure C-26 Interface to define a new graph from a table view](image)

Use the Graph Definition's General tab to define the graph description (the graph’s title), the type of graph, and the bars per page (Figure C-27).

![Figure C-27 Graph Definition initial menu](image)
Use the X-axis tab to select the fields for the X axis labels. Up to three can be specified (Figure C-28).

![Graph Definition X axis menu](image)

**Figure C-28  Graph Definition X axis menu**

Use the Primary Y-axis tab to define the field(s) from the data view to be displayed on the graph. Each field represents part of a bar on a stacked bar graph and can have a different color and customized description. See Figure C-29 on page 222. On this menu you can fill in:

1. The description: This is the graph’s title.
2. The Scale: The largest and smallest values to show on this axis.
3. The fields. Do the following for each field to be graphed:
   a. Select the field from the drop down list. The field description is automatically filled in using the field’s description.
   b. Click the **Change...** button to the right of Color. This brings up a menu used to select the field’s color.
   c. Optionally, set the **Border color** and **Border width**.
4. Click **Add Field** to include the field in the graph.

After a field has been added you can change its definition or remove it from the graph by selecting it and using the **Update** or **Remove** buttons.
Figure C-29  Graph Definition Primary Y axis menu

Use the Secondary Y-axis tab’s menu (Figure C-30) to define a secondary Y-axis.

Follow the same procedures for this axis as those specified for the primary Y-axis on the previous menu.

Figure C-30  Graph Definition Secondary Y axis menu
At any point, you can select OK to view the graph. See Figure C-31, which graphs the data from Figure C-26 on page 220.

**Figure C-31**  Graph of Program and Instruction Elapsed time and CPU use

Using the view in Figure C-32, right-click in the graph view to get a Graph Definition submenu.

**Figure C-32**  Graph Definition menu selection
The options are:

**Copy**  Copy the graph view to the clipboard.

**Save As**  Save the graph view to disk as a JPG file.

**Preferences**  Bring up the iDoctor preferences menu.

**Graph Definition**  Select a **Graph Definition** menu to change the graph. This is the menu you use to save the graph definition.

**Query Definition**  Change the graph’s underlying query definition.

**Create Alternate view**  There are two options here:

- Show the table data view.
- Show the table data view with the SQL query window.

**Properties**  Same as the table view properties, for example, show the properties of the Query, Analysis, or Collection.
PEX Stats user-defined queries

This appendix shows some SQL queries that might be useful in various performance study situations. All of the following, except for the SQL for Figure D-13 on page 232, use Hierarchical data. Note that in some of the queries the times are presented in seconds instead of the standard PEX Stats use of microseconds.

The addition of highly efficient job specific analysis in V5R3 PEX Stats Flat provides powerful new ways to look at the data. While some of these views have not been incorporated into PEX Stats Analysis programs as of this writing, building and using them as user-defined queries that can be run over the PEX Stats analysis output data files provides equivalent ease of use, function, and information.
Creating a special query

Decide what data and data organization you need. Then build a query using these steps as a guideline:
1. Bring up the Query Definition menu.
2. Using the Field Selection tab, select the fields you want, where you want them in the view and, if needed, define any new fields.
3. Using the Record Selection tab, set the Record Selection parameters.
4. Using the Sort By tab, set the Sort key(s): order of the key, the key’s field name, and ascending or descending sort (ascending is the default).
5. Using the Group By tab, set the Group By criteria. You can build a Having statement if needed in the Group By window's lower pane.
6. Save the Query as a user defined query.

Entering your own SQL

You can build your own SQL Select statement with iDoctor’s New SQL Query function. There is more information about how to do this in “iDoctor's New SQL Query view interface” on page 217.

One way to enter a SQL statement is to copy it to the clipboard from:
- A data view's Properties → Query menu
- An i5/OS STRSQL window
- An example in this book

and paste it into an iDoctor New SQL Query window.
1. To begin, select iDoctor’s Data Viewer icon to get a new, empty data view.
2. Select iDoctor's New SQL Query icon,
3. Select the system on which you want the SQL to run,
4. Key in or copy and paste the SQL Select statement shown in Figure D-3 on page 227 or some other source, such as a System i STRQRY window’s SQL statement line into the SQL Query view (the top) part of the window in Figure D-1 on page 227.
5. If necessary, change the SQL FROM clause's libname to your data library name.
6. Run the query by either right-clicking the view and select the menu’s top line (Execute F4) or just press F4.
7. After it runs, the result appears in the bottom part of the SQL Query View window in Figure D-2 on page 227.
Appendix D. PEX Stats user-defined queries

Figure D-1   New SQL Query -- entering the SELECT statement

To get to here:
1. Open a new Data Viewer.
2. Select the New SQL Query icon.
3. Paste or key in your SQL statement.

Figure D-2   New SQL Query: Execute / F4 results

This view shows the activity in the REFID range 664 to 683. It is used to compare the costs of invoking a QCL... program to the function performed by the QCL.... program

Figure D-3 shows the SQL for Figure D-2.

Figure D-3   SQL for Figure D-2

```
SELECT CALLLVL, LIBNAM, PGMNAM, MSCPXM, CALLCOUNT, CALLMADE, CALLMCPE, INELPM, INCPUUS, CUELPUS, CUCPUUS, REFID FROM libname/G_STATSH WHERE REFID BETWEEN 664 AND 683 ORDER BY REFID ASC
```
Extract database, query, and SQL usage information

Figure D-4 shows how to extract database, query, and SQL usage information.

To see a collection wide summary of database program activity, use the query in Figure D-5 to select information about Data Base, SQL, Query, and Journal components.

The query shows the program usage and costs by selecting data for all programs whose name starts with QDB, QSQ, QQQ, or QJO. The sort key is CUMELAPSED_SEC (Cumulative Elapsed Seconds) descending.

```
SELECT LIBNAM, PGMNAM, SUM(CALLCOUNT) AS TOTTIMESCALLED, 
  dec(SUM(CUELPUS)/1e6,12,6) AS CUMELAPSED_SEC, 
  dec(SUM(INELPUS)/1e6,12,6) AS TOTINL_SEC, 
  dec(SUM(CUCPUUS)/1e6,12,6) AS TOTCUMCPU_SEC, 
  dec(SUM(INCPUUS)/1e6,12,6) AS TOTINLCPU_SEC, 
  SUM(CUS) AS TOTCUMSYNIO FROM libname/G_STATSH WHERE SUBSTR(PGMNAM,1,3) IN ('QDB', 'QSQ', 'QQQ', 'QJO') GROUP BY LIBNAM, PGMNAM ORDER BY CUMELAPSED_SEC DESC
```

Figure D-5  SQL for Extract database, SQL, and Query usage information in Figure D-4
Extract data base logical read, write and update usage information

Figure D-6 and Figure D-7 shows how to extract data base logical read, write, and update usage information.

**Figure D-6**  Data base Logical DIO: get, put, and update overall and per call costs

```sql
SELECT LIBNAM, PGMNAM,
       SUM(CALLCOUNT) AS TotalTimesCalled,
       dec(SUM(CUELPUS)/1e6,12,6) AS PgmCumElapsedSec,
       dec(SUM(CUCPUUS)/1e6,12,6) AS PgmTotCumCPUsec,
       dec(SUM(CUELPUS)/SUM(CALLCOUNT)/1e6,12,6) AS PgmElpSecPerCall,
       SUM(CUCPUUS)/SUM(CALLCOUNT) AS PgmCPUsecPerCall
FROM   libname/G_STATSH
WHERE   SUBSTR(PGMNAM,1,4) IN ('QDBG', 'QDBP', 'QDBU')
GROUP BY LIBNAM, PGMNAM
ORDER BY PgmCumElapsedSec DESC
```

**Figure D-7**  Data base get, put, and update cost per call
Total cumulative synchronous data base reads by program

This query (Figure D-8 and Figure D-9) selects only application programs, that is, those programs whose names do not start with the letter Q or do not start with a blank.

```
SELECT QTSJNM, QTSJUS, LIBNAM, PGMNAM, sum(qstinv) AS SUMTIMESCALLED, sum(qcisdr) AS CUMSYNDBRD, sum(qciadr) AS SUMASYNCDBREAD FROM LIBNAME/G_STATSFJB WHERE PGMNAM NOT LIKE 'Q%' AND PGMNAM NOT LIKE ' %' GROUP BY QTSJNM, QTSJUS, LIBNAM, PGMNAM having sum(qstinv)>100 and sum(qcisdr) > 0 and sum(qciadr) > 0 ORDER BY SUMTIMESCALLED DESC
```

Figure D-9  SQL for Total cumulative synch data base reads by program in Figure D-8
Show summary activity for all Hierarchical collection rows

Figure D-10 and Figure D-11 shows the summary activity for all Hierarchical collection rows.

![Figure D-10 All Program, procedure and MI Instruction activity](image)

SELECT LIBNAM, PGMNAM, PRCNAM, MICPXNM, SUM(CALLCOUNT) AS TCALLCOUNT, SUM(CALLMADE) AS TCALLMADE, SUM(CALLMICPX) AS TCALLMICPX, SUM(CUELPUS) / 1e6 AS TCUMELPCPUSEC, SUM(INELPUS) / 1e6 AS TINELPSEC, SUM(CUCPUUS) / 1e6 AS TCUMCPUSEC, SUM(INCPUUS) / 1e6 AS TINCPUSEC, SUM(INPCPUUS) AS TINPCTCPUSEC, SUM(INCPUPI) / 1e6 AS TINCPUSECPERCALL, SUM(INPS) AS TINPCTSYNCHIO, SUM(INPA) AS TINPCTASYNCHIO, SUM(INPW) AS TINPCTWAITONIO, SUM(INW) AS TWAITONIO, SUM(INSSWA) AS TINSYNCHIOWAIT, SUM(INSPWA) AS TINIOPENDWAIT, SUM(SINS) AS TINSYNCHIO, SUM(SIN) AS TINIOPEND, SUM(SINPS) AS TINPCTSYNCHDBIO, SUM(SINPSN) AS TINPCTASYNCHDBIO, SUM(SINPAD) AS TINPCTSYNCHDBREAD, SUM(SINPSSWA) AS TINPCTSYNCHIOWAIT, SUM(SINPADR) AS TINPCTASYNCHDBREAD, SUM(SINPADW) AS TINPCTASYNCHDBWRITE FROM library/G_STATSH WHERE refid >= 0 GROUP BY LIBNAM, PGMNAM, PRCNAM, MICPXNM ORDER BY LIBNAM ASC, PGMNAM ASC, PRCNAM ASC, MICPXNM ASC

![Figure D-11 SQL for All Program, procedure, and MI Instruction activity in Figure D-10](image)
MI instruction activity: Hierarchical data

Figure D-12 and Figure D-13 shows the MI instruction activity.

![Figure D-12 MI Instruction activity](image)

**Figure D-12**  MI Instruction activity

```sql
SELECT MICPXNM, CALLCOUNT, INELPUS, INCPUUS, INPCPUUS, INCPUPI, INS, INPS, INA, INPA, INSD, INPSD, INPSDR, INADR, INPADR, INSNR, INSPWA, INSDW, INADW, INSNW, INANW, INSSWA, INSN, INPSN, INAD, INPAD, INAN
FROM libname/G_STATSH WHERE MICPXNM LIKE '*%' ORDER BY INCPUPI DESC
```

**Figure D-13**  SQL for MI Instruction activity: Hierarchical data

Special Collection: DB Full Open counts by program

This section describes the use of a PEX object called a PEX Filter.

PEX Filters are used to selectively include or omit certain information from a PEX collection. It is applicable for each PEX collection type (Profile, Trace, and Stats).

This discussion's scope is limited to user programs that issue full data base open requests.

The filter collects an i5/OS event called *DBOPEN (a PEX Data Collection's event type 12, subtype 2) only for full data base Opens and it excludes *DBCLOSE events. The filter causes recording of only application program names for the PEX Stats collection of MI ENTRY/EXIT user program Call/Return events. It also excludes (that is, filters out) all programs in the QSYS library.
The steps

The collection requires a special PEX Filter and PEX Definition. To build them, go to an i5/OS command line and run the following commands:

```plaintext
ADDPEXFR FTR(DBOPENONLY) PGMFTR(*NE ((QSYS/*ALL *ALL *ALL *PGM))) USRDFN FTR(*EQ 12 2 144 *CHAR (O))) TEXT('Include QDBOPEN but not QDBCLOSE')

and

ADDPEXDFN DFN(DBOPENONLY) JOB((*ALL/*ALL/*ALL)) TASK(*ALL) MRGJOB(*NO) PGMBKTEVT(*MENTRYEXIT) SLEVT(*YES) OSEVT((*DBOPEN 1)) TEXT('Put counts of full db file opens into counter 1')
```

1. Run your test, either from iDoctor’s Start Collection menu or use the command line and run the command:

   ```plaintext
   STRPEX SSNID(FULLOPENS) DFN(DBOPENONLY) FTR(DBOPENONLY)
   ```

2. Run your test.

3. To end the collection, use the command ENDPEX to end the collection and save the data.

   **Note:** In no case should you mix PEX starting and stopping methodology by using the STRPEX command and stopping it with iDoctor or by starting it with iDoctor and ending it with ENDPEX. Do not do it. It is not checked for, but can compromise your data integrity.

4. Go to iDoctor and run a PEX Stats analysis over the collection.

5. In iDoctor, use the New SQL Query window and run the SQL shown in Figure D-14 to retrieve and view the data.

   ```plaintext
   SELECT LIBNAM, PGMNAM, MICPXNM, MODNAM, PROC_SHORT, QSTINV05, QSTINV01, QSTCCT01, QSWIO101, INLCPU01, INLELP01, CUMELP01 FROM G_STATSFPG ORDER BY QSWI0101 DESC
   ```

   *Figure D-14  SQL to show the Data Base Full Open count by User Program and Library*

6. The output appears as in Figure D-15. Note the Inline Counter 1 and the Program name and library information.

*Figure D-15  DB Full Open count by User Program and Library*
Program and MI Complex
Instructions summary and Stats’
analysis file descriptions
i5/OS programs and MI Complex Instructions short descriptions

This appendix contains information about some i5/OS programs and MI Complex Instructions, what functions they perform and, in some cases, PEX Stats analysis considerations that should be taken into account.

i5/OS programs

In some cases, the i5/OS program names more easily relate to program function than do MI Complex Instructions. The following list summarizes the activity of some of the more commonly used i5/OS programs.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QC2IO</td>
<td>This is an ILE C program that interfaces between the user application program and i5/OS data management (for example, database and workstation I/O operations).</td>
</tr>
<tr>
<td>QCA*</td>
<td>Command analyzer modules. These often represent CL commands used instead of HLL programs.</td>
</tr>
<tr>
<td>QCADRV</td>
<td>This module handles all CL commands and, in so doing, calls several other i5/OS modules. There is a one-to-one relationship between the number of calls of this module and the number of CL commands used in a CL program, run in a HLL program, and entered interactively. This module calls all QCA* modules.</td>
</tr>
<tr>
<td>QCAPOS</td>
<td>Called by the QCADRV module.</td>
</tr>
<tr>
<td>QCARULE</td>
<td>Called by the QCADRV module.</td>
</tr>
<tr>
<td>QCLRTVDA</td>
<td>Retrieve Data Area.</td>
</tr>
<tr>
<td>QCPEXCON</td>
<td></td>
</tr>
<tr>
<td>QCPEX0FL</td>
<td>(CPYF) (Copy File) Do not do a lot of big copies during prime time or heavy workload periods.</td>
</tr>
<tr>
<td>QDB*</td>
<td>Database module.</td>
</tr>
<tr>
<td>QDBCLOSC</td>
<td>CL program CLOF operation’s command processing program.</td>
</tr>
<tr>
<td>QDBEXDFI</td>
<td>(CRTPF, DSPFFD) (Create Physical file, Display Physical File Description and other commands). High use can occur in object distribution applications. Often may have high ASM activity related to it.</td>
</tr>
<tr>
<td>QDBGETKY</td>
<td>Accesses a file for input or update using a key for random access. An example is chaining in RPG.</td>
</tr>
<tr>
<td>QDBGETM</td>
<td>Sequential reads using blocking for input only.</td>
</tr>
<tr>
<td>QDBGETMQO</td>
<td>The SQE(SQL Query Engine) equivalent of QDBGETM.</td>
</tr>
<tr>
<td>QDBGETSQ</td>
<td>Sequential reads, with each record retrieved individually. In this instance, a RPG program is not using blocking.</td>
</tr>
<tr>
<td>Program</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QDBOPEN</td>
<td>Data base file Full Open and Close. Keep the total times called of these two below 100 per second to maintain more stable throughput and response time.</td>
</tr>
<tr>
<td>QDBCLOSE</td>
<td>Command Language commands OPNDBF and CLSDBF command processing programs (CPP). Normally low rates.</td>
</tr>
<tr>
<td>QDBOPENC</td>
<td>CL Program OPNDBF operation's command processing program.</td>
</tr>
<tr>
<td>QDBPUT</td>
<td>Used to add or write non-blocked records to a database file.</td>
</tr>
<tr>
<td>QDBPUTM</td>
<td>Applies to the blocking in output files only. QDBCLOSE Full Closes are done against database files.</td>
</tr>
<tr>
<td>QDBSECOFR</td>
<td>Calls to this program occur when the user has authority to a view or access path but does not have authority to the underlying physical. One reason could be that there are other columns in the physical that the user is not supposed to see. There is a V5R4 PTF to set the authority in the based-on file's pointer to avoid the authority exception and the calls to QDBSECOFR. This requires recreation of the based-on logical file (or view).</td>
</tr>
<tr>
<td>QDBSOPEN</td>
<td>Shared opens done against database files.</td>
</tr>
<tr>
<td>QDDCLF</td>
<td>(CRTLF) Create Logical File: If due to QQQ..., then the files and their usage probably need to be evaluated.</td>
</tr>
<tr>
<td>QDDCPF</td>
<td>(CRTPF) Create Physical File: Occasional use is fine, while consistent use may be an indication of lots of scratch file creation/deletion.</td>
</tr>
<tr>
<td>QDMCLOSE</td>
<td>The number of calls of this module is equal to the total number of all closes of all files except spooled files.</td>
</tr>
<tr>
<td>QDMCOPEN</td>
<td>The number of calls of this module is equal to the total number of file opens (excluding spooled files).</td>
</tr>
<tr>
<td>QLICRDUP</td>
<td>(CRTDUPOBJ) Create Duplicate Object: Same comments as copy file and create physical file.</td>
</tr>
<tr>
<td>QLN*</td>
<td>This is an ILE COBOL run time support module.</td>
</tr>
<tr>
<td>QLNRFSEQ</td>
<td>This is an ILE COBOL run time program that interfaces between the user application program and i5/OS data management (for example, database and workstation I/O operations).</td>
</tr>
<tr>
<td>QLRMAIN</td>
<td>Run time program initialization for OPM COBOL programs.</td>
</tr>
<tr>
<td>QMH*</td>
<td>Message handling modules, such as record not found or added record already existing.</td>
</tr>
<tr>
<td>QMHSNJMQ</td>
<td>Puts messages to the job log. When high, discover the cause and attempt to eliminate the problem. More than 1% of CPU time warrants further investigation.</td>
</tr>
<tr>
<td>Module</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QQQ*</td>
<td>Modules used for QUERY and SQL. Use of traditional HLL programs with DB I/O and access paths instead of these may improve system performance.</td>
</tr>
<tr>
<td>QQQIMPLE</td>
<td>Query initialization. Usually has high disk I/O and CPU use due to building a database index. Is called as a result of decisions made by QQQOPTIM. Used to decide how to run the query. Sometimes it calls the Query Optimizer.</td>
</tr>
<tr>
<td>QQQOPTIM</td>
<td>Query optimizer. Decides how to run a query, SQL, or OPNJQRYF and whether or not to build an index. Frequent use is an indication of possibly incorrect or incompatible query or file properties. Changes in each release may give totally different results between releases, or as a result of access paths being added or taken away from a physical file.</td>
</tr>
<tr>
<td>QRCVDTAQ</td>
<td>Receive or Wait for a Data Queue message.</td>
</tr>
<tr>
<td>QRGXINIT</td>
<td>Program initialization for OPM RPG. Low use is OK, high use might be a problem.</td>
</tr>
<tr>
<td>QRNXINIT</td>
<td>Program initialization for ILE RPG. Low use is OK, high use might be a problem.</td>
</tr>
<tr>
<td>QRNXIO</td>
<td>An ILE RPG Run Time Support program that interfaces between the user application program and i5/OS data management (for example, database and workstation I/O operations).</td>
</tr>
<tr>
<td>QSF*</td>
<td>Subfile module.</td>
</tr>
<tr>
<td>QSFCRT</td>
<td>Each time this module is called, a subfile (part of a display file) is created.</td>
</tr>
<tr>
<td>QSFGET</td>
<td>Each time this module is called, a record out of a subfile is read from a program.</td>
</tr>
<tr>
<td>QSFPUT</td>
<td>Writing of subfile records to the subfile. This module is called every time you write a record to a subfile from your program.</td>
</tr>
<tr>
<td>QSNAPI</td>
<td>ILE C generalized screen I/O interface module.</td>
</tr>
<tr>
<td>QSNDDTAQ</td>
<td>Send Data Queue message.</td>
</tr>
<tr>
<td>QSP*</td>
<td>Spool component.</td>
</tr>
<tr>
<td>QSPOPEN</td>
<td>Spool file Full Open. Spooled files have two opens: one against the device file and one against the spooled file member where the physical data actually resides. The times called count for this module equals the number of spooled files opened.</td>
</tr>
<tr>
<td>QSR*</td>
<td>i5/OS Save Restore component. Indicates possible save/restore activity that runs in SLIC tasks, not in the user's job.</td>
</tr>
<tr>
<td>QTE*</td>
<td>Test Component. Program may be in debug mode.</td>
</tr>
<tr>
<td>QUI*</td>
<td>These are system user interface modules. These modules do most of the screen formatting for system commands and other functions.</td>
</tr>
<tr>
<td>QUIINMGR</td>
<td>User Interface Manager Interaction program that interfaces to workstation data management on behalf of system functions.</td>
</tr>
<tr>
<td>Module</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QWCCCHVC</td>
<td>Change Data Area.</td>
</tr>
<tr>
<td>QWCCCHVC</td>
<td>Change data area by a job/thread.</td>
</tr>
<tr>
<td>QWCCRCRC</td>
<td>Reclaim resources. Sometimes the “normal” result of program termination, sometimes an explicit use of the RCLRSC command. (Reclaim Resources). Investigate cause of high Times Called.</td>
</tr>
<tr>
<td>QWP*</td>
<td>Work station printer support modules.</td>
</tr>
<tr>
<td>QWPPTFLD</td>
<td>This module is used in conjunction with print files. It does formatting of print output for SNA/SCS protocols, and can be a high user of CPU time or cumulative CPU time, especially if 3270 device emulation is being used.</td>
</tr>
<tr>
<td>QWPPUT</td>
<td>This module usually calls the QWPPTFLD module, and as such writes records to a print file.</td>
</tr>
<tr>
<td>QWS*</td>
<td>Work station display support modules.</td>
</tr>
<tr>
<td>QWSGET</td>
<td>Each call of this module represents the read of one display file format, or the second part of a RPG EXFMT. The number of calls of this module equals approximately the number of times the Enter key is pressed.</td>
</tr>
<tr>
<td>QWSOPEN</td>
<td>Full Opens for work station display files.</td>
</tr>
<tr>
<td>QWSPUT</td>
<td>Sending of formats to the screen. If Cumulative CPU for this module is high, you may see improved system performance by adding deferred writes. The call of this module represents the write of a format of a display file from a program to a screen, or the use of an RPG EXFMT instruction.</td>
</tr>
<tr>
<td>QWSSFLCT</td>
<td>This module is called when the subfile control record is written.</td>
</tr>
<tr>
<td>QWVCCDLA</td>
<td>Part of Delete Activation Group.</td>
</tr>
<tr>
<td>QZDA*</td>
<td>These are i5/OS Host server (database, file, and so on) modules.</td>
</tr>
</tbody>
</table>
### MI Complex Instructions

This list summarizes some of the MI Complex Instructions and, in some cases, how they relate to i5/OS modules. Unless you are familiar with MI Complex Instructions, it is easier to determine what a program is doing by referring to the i5/OS modules.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTBPGM</td>
<td>Activate Bound Program: Low use OK, high use might be a problem. Look for the MI program (up the Hierarchical data call path.)</td>
</tr>
<tr>
<td>ACTBPGM2</td>
<td>Activate Bound Program 2: Handles the newer larger activation marks. It can occur if someone is creating *NEW activations a lot. It can be very expensive depending on how many service programs are activated.</td>
</tr>
<tr>
<td>ACTCR</td>
<td>Activate Cursor. This MI Complex Instruction is used for full DB opens (includes spooled files opens from both intercept time and the printer writer).</td>
</tr>
<tr>
<td>ACTPG</td>
<td>Activate Program. A high count can be the result of improper program structuring or ILE techniques. Look for the same MI program (up the Hierarchical data call path.</td>
</tr>
<tr>
<td>COMMIT</td>
<td>Commit changes to the data base: Interface with DB support to establish a commit point and write (force) any pending journal entries to the journal receiver.</td>
</tr>
<tr>
<td>CRTBPGM</td>
<td>Create Bound Program. Usually means a compile operation was done during the data collection. Indicates program maintenance occurring on the system. May be okay, should be investigated.</td>
</tr>
<tr>
<td>CRTCB</td>
<td>Create Commit Block: Part of setting up a commitment control environment for the file.</td>
</tr>
<tr>
<td>CRTCRR</td>
<td>Create Cursor. This is used for full DB opens.</td>
</tr>
<tr>
<td>CRTDMPS</td>
<td>Create Dump Space. Cause could be due to programming errors, job termination, or it could be part of the application's design.</td>
</tr>
<tr>
<td>CRTDOBJ</td>
<td>Create Duplicate Object: One per full DB file open. Full Open's high use of object create and destroy can affect the whole system environment and performance. Sometimes there are corresponding ACTCR (Activate Cursor) Instructions</td>
</tr>
<tr>
<td>CRTDS</td>
<td>The creation of any data space (DS) is the simplest form of a database object. It is used for PF members.</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CRTDSINX</td>
<td>Create Data Space index. Used to build an index over data base file(s).</td>
</tr>
<tr>
<td></td>
<td>Can be very expensive. Stats shows the elapsed and CPU time cost of creation.</td>
</tr>
<tr>
<td></td>
<td>Does not show the time cost of delaying other jobs that need to update the physical file during the index build.</td>
</tr>
<tr>
<td></td>
<td>Can be very time consuming. Investigate the system and possible job override setting for QQRYDEGREE.</td>
</tr>
<tr>
<td>CRTINX</td>
<td>Create Independent Index: Add a stand-alone index to the system. See if there are corresponding DESINX instructions.</td>
</tr>
<tr>
<td>CRTPG</td>
<td>Program creation. This MI Complex Instruction usually means a compile operation was done during the data collection. Indicates program maintenance occurring on the system. May be okay, should be investigated.</td>
</tr>
<tr>
<td>CRTS</td>
<td>Create Space: Add an object to the system. Has many uses. Investigate high use, especially if there are a lot of corresponding DESS (has the same or very close times called value).</td>
</tr>
<tr>
<td>DEACTBM</td>
<td>Deactivate Bound Module.</td>
</tr>
<tr>
<td>DEACTPG</td>
<td>Deactivate Program. High use of this from a specific program can be the cause of lots of Full Opens during program initialization. It is often the result of using a RPG Program RETRN with LR ON, a COBOL STOP RUN instead of EXIT or GO BACK, or C++ creating procedure indicating return and destroy the program activation. Also can be caused by an ILE program using CALLER *NEW and returning a lot. This combination is especially severe.</td>
</tr>
<tr>
<td>DECOMMIT</td>
<td>Remove committed changes from data base -- destroy the commit point and remove the journal changes from the journal and rollback the changes made to the data base file. This can be very costly if there is a high DECOMMIT count. Investigate.</td>
</tr>
<tr>
<td>DEQ</td>
<td>Dequeue a message, similar to *DEQWAIT, except that *DEQ never waits.</td>
</tr>
<tr>
<td>DEQWAIT</td>
<td>Dequeue, and wait if necessary, for a message. This is used to wait for workstation I/Os, most (except stream I/O operations) communications I/Os, or data queue dequeues.</td>
</tr>
<tr>
<td>DESAGP</td>
<td>User program Destroy Activation Group. Investigate high use.</td>
</tr>
<tr>
<td>DESCB</td>
<td>Destroy Commit Block: Removing the commitment control environment for the file. Lots of these and CRTCB in a job should be investigated.</td>
</tr>
<tr>
<td>DESCR</td>
<td>Destroy Cursor. This is used for full DB closes.</td>
</tr>
<tr>
<td>DESDMPS</td>
<td>Destroy Dump Space (see CRTDMPS above).</td>
</tr>
<tr>
<td>DESDS</td>
<td>Destroy Data Space.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DESS</td>
<td>Destroy Space: Remove an object from the system.</td>
</tr>
<tr>
<td>ENSOBJ</td>
<td>Ensure object. May be associated with using FEOD in an RPG program to move a new record out of the job's internal work area (the ODP buffer) and add it as new record to the data base. May also results from use of FRCRATIO(1) on the OVRDBF command to force each changed record from the job into the data base. Both FEOD and FRCRATIO(1) force a single record synchronous write to the disk and can often cause a single record write to the database journal. If caused by use of FEOD, change the application program to FEOD(N) to get the record added to the file, but allow the system to synchronously write the single record to disk. FRCRATIO is unnecessary on the System i and impacts performance. If ENSOBJ shows high non-DB record writes, more investigation with Job Watcher or PEX Disk I/O Trace is necessary to determine what jobs and objects are being ensured. Also used to ensure changes to an object by immediately writing them to auxiliary storage. Data area updates are handled with these.</td>
</tr>
<tr>
<td>ESTDSIKR</td>
<td>Estimate Data Space Index Key Range. It is used often in SQL/OPNQRYF/Query, and is used to guide how the query will be run. It may result in CRTDSINX usage.</td>
</tr>
<tr>
<td>INSDSEN</td>
<td>Insert data space entry and MI Complex Instruction for the random write (add) of a new record.</td>
</tr>
<tr>
<td>INSSDSE</td>
<td>Insert sequential data space entry MI Complex Instruction for the QDBPUTM module.</td>
</tr>
<tr>
<td>MATPTR</td>
<td>Materialize the pointer to an object already found and authorized to.</td>
</tr>
<tr>
<td>RETDSEN</td>
<td>MI Complex Instruction for the QDBGETSQ module - Unblocked DB reads.</td>
</tr>
<tr>
<td>RETSDSE</td>
<td>MI Complex Instruction for the QDBGETM module - Blocked DB reads.</td>
</tr>
<tr>
<td>RSLVSP</td>
<td>Resolve system pointer caused by Full Opens, commands, message handling, and late bound calls (calls using a variable).</td>
</tr>
<tr>
<td>SETCR</td>
<td>Set Cursor Index positioning.</td>
</tr>
<tr>
<td>UPDSEN</td>
<td>Update of all records. The relationship between this and *RETDSEN shows the percentage of read records actually updated.</td>
</tr>
<tr>
<td>WAITEVT</td>
<td>Work station I/O wait in System/36™ environment.</td>
</tr>
</tbody>
</table>
Examples of i5/OS Module Relationships

Examples of relationships between i5/OS modules include the following:

- Database blocking factors, and random versus sequential accessing can be determined (especially for a dedicated batch run) by looking at the call counts for QDBGETSQ, QDBGETM, QDBPUT, and QDBPUTM.

- If you know how many records were read or written by a job, you can determine the HLL blocking factor by comparing the number of records to the number of calls of QDBGETM and QDBPUTM. These are called once per block. If you see QDBGETSQ or QDBPUT instead, then no HLL program blocking occurred. This indicates that SEQONLY(*YES number-of-records) on the Override Database File (OVRDBF) command either needs to be used, or it is used, but for some reason the system is ignoring the function. Check the job for a diagnostic message. Similar logic can be applied to i5/OS and LIC blocking.

- The total number of Full Opens = QDBOPEN + QWSOPEN + QSPOPEN. Therefore, the ratio of full to shared opens is approximately: (number of Full Opens) / ((QDMCOPEN + QSPOPEN) - number of Full Opens).

- The ratio of full DB opens to shared opens is QDBOPEN/QDBSOPEN. This information, combined with the total CPU for opens and closes (CCPU% - Cumulative CPU Utilization percentage for QDMCOPEN and QDMCLOSE) and the knowledge that shared opens cost approximately 90% less than a Full Open, allows you to calculate how much leverage you have in doing more shared opens or eliminating opens and closes by leaving programs active or by using group jobs.

- A large number for QSPOPEN without many spooled files actually being produced indicates programs opening print files and not writing anything to them (an example is a debug file). In other words, an open and close of a spooled file is being done for no reason. Open a spooled file only when you need it. If QSPOPEN appears frequently as a result of running an interactive application, consider having a portion of the application partitioned to run in batch.

---

| DBGINT  | Blocked instructions used by the system debugger and Java debugger and should not show up in production code. Ensure changes to object by immediately writing them to auxiliary storage.
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| DBGJV   | Data area updates are handled with these. |
PEX Stats data analysis output files field names: V5R3 / V5R4

Here we discuss PEX Stats data analysis output files field names.

Stats Flat Program file G_STATSFPG

<table>
<thead>
<tr>
<th>Field name</th>
<th>Field description</th>
<th>Output buffer position</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBNAM</td>
<td>Library name 1</td>
<td></td>
</tr>
<tr>
<td>PGMNAM</td>
<td>Program name 11</td>
<td></td>
</tr>
<tr>
<td>MICPXNM</td>
<td>MI Complex Instruction 21</td>
<td></td>
</tr>
<tr>
<td>MODNAM</td>
<td>Module name 41</td>
<td></td>
</tr>
<tr>
<td>PROC_SHORT</td>
<td>Procedure short name 56</td>
<td></td>
</tr>
<tr>
<td>QSTINV05</td>
<td>Number of threads 71</td>
<td></td>
</tr>
<tr>
<td>QSTINV01</td>
<td>Times called 78</td>
<td></td>
</tr>
<tr>
<td>QSTCCT01</td>
<td>Calls made 99</td>
<td></td>
</tr>
<tr>
<td>QSTXCT01</td>
<td>MI complex instruction count 120</td>
<td></td>
</tr>
<tr>
<td>INLCPU01</td>
<td>Inline CPU usecs 141</td>
<td></td>
</tr>
<tr>
<td>CUMCPU01</td>
<td>Cumulative CPU usecs 165</td>
<td></td>
</tr>
<tr>
<td>INLELP01</td>
<td>Inline elapsed usecs 189</td>
<td></td>
</tr>
<tr>
<td>CUMELLP01</td>
<td>Cumulative elapsed usecs 213</td>
<td></td>
</tr>
<tr>
<td>QIIISDR01</td>
<td>Inline synchronous DB reads 237</td>
<td></td>
</tr>
<tr>
<td>QIIISNR01</td>
<td>Inline synchronous non-DB reads 249</td>
<td></td>
</tr>
<tr>
<td>QIIISDW01</td>
<td>Inline synchronous DB writes 261</td>
<td></td>
</tr>
<tr>
<td>QIIISNW01</td>
<td>Inline synchronous non-DB writes 273</td>
<td></td>
</tr>
<tr>
<td>QIIADR01</td>
<td>Inline asynchronous DB reads 285</td>
<td></td>
</tr>
<tr>
<td>QIIANR01</td>
<td>Inline asynchronous non-DB reads 297</td>
<td></td>
</tr>
<tr>
<td>QIIADW01</td>
<td>Inline asynchronous DB writes 309</td>
<td></td>
</tr>
<tr>
<td>QIIANW01</td>
<td>Inline asynchronous non-DB writes 321</td>
<td></td>
</tr>
<tr>
<td>QSWI0101</td>
<td>Inline software counter 1 333</td>
<td></td>
</tr>
<tr>
<td>QSWI0201</td>
<td>Inline software counter 2 354</td>
<td></td>
</tr>
<tr>
<td>QSWI0301</td>
<td>Inline software counter 3 375</td>
<td></td>
</tr>
<tr>
<td>QSWI0401</td>
<td>Inline software counter 4 396</td>
<td></td>
</tr>
<tr>
<td>QIIIPWA01</td>
<td>Inline I/O pending waits 417</td>
<td></td>
</tr>
<tr>
<td>QIISWA01</td>
<td>Inline synchronous I/O waits 429</td>
<td></td>
</tr>
<tr>
<td>QSWC0101</td>
<td>Cumulative software counter 1 441</td>
<td></td>
</tr>
<tr>
<td>QSWC0201</td>
<td>Cumulative software counter 2 462</td>
<td></td>
</tr>
<tr>
<td>QSWC0301</td>
<td>Cumulative software counter 3 483</td>
<td></td>
</tr>
<tr>
<td>QSWC0401</td>
<td>Cumulative software counter 4 504</td>
<td></td>
</tr>
<tr>
<td>QCISDR01</td>
<td>Cumulative synchronous DB reads 525</td>
<td></td>
</tr>
<tr>
<td>QCISNR01</td>
<td>Cumulative synchronous non-DB reads 537</td>
<td></td>
</tr>
<tr>
<td>QCISDW01</td>
<td>Cumulative synchronous DB writes 549</td>
<td></td>
</tr>
<tr>
<td>QCISNW01</td>
<td>Cumulative synchronous non-DB writes 561</td>
<td></td>
</tr>
<tr>
<td>QCIAADR01</td>
<td>Cumulative asynchronous DB reads 573</td>
<td></td>
</tr>
<tr>
<td>QCIANR01</td>
<td>Cumulative asynchronous non-DB reads 585</td>
<td></td>
</tr>
<tr>
<td>QCIAADW01</td>
<td>Cumulative asynchronous DB writes 597</td>
<td></td>
</tr>
<tr>
<td>QCIANW01</td>
<td>Cumulative asynchronous non-DB writes 609</td>
<td></td>
</tr>
<tr>
<td>QCPLNM</td>
<td>Long MI complex description 621</td>
<td></td>
</tr>
<tr>
<td>PGMDSC</td>
<td>Program description 671</td>
<td></td>
</tr>
<tr>
<td>QPRPNM</td>
<td>Procedure name 741</td>
<td></td>
</tr>
</tbody>
</table>
### Stats Flat Job file G_STATSFJB

<table>
<thead>
<tr>
<th>Field name</th>
<th>Field description</th>
<th>Output buffer position</th>
</tr>
</thead>
<tbody>
<tr>
<td>QTSJNM</td>
<td>Job name 1</td>
<td>1</td>
</tr>
<tr>
<td>QTSJUS</td>
<td>Job user 11</td>
<td>11</td>
</tr>
<tr>
<td>QTSJNB</td>
<td>Job number 21</td>
<td>21</td>
</tr>
<tr>
<td>QTSTHI</td>
<td>Job thread id 27</td>
<td>27</td>
</tr>
<tr>
<td>QTSTIF</td>
<td>Initial thread Y or N 35</td>
<td>35</td>
</tr>
<tr>
<td>LIBNAM</td>
<td>Library name 36</td>
<td>36</td>
</tr>
<tr>
<td>PGMNAM</td>
<td>Program name 46</td>
<td>46</td>
</tr>
<tr>
<td>MICPXNM</td>
<td>MI complex instruction 56</td>
<td>56</td>
</tr>
<tr>
<td>MODNAM</td>
<td>Module name 76</td>
<td>76</td>
</tr>
<tr>
<td>PRCNAMST</td>
<td>Procedure name 91</td>
<td>91</td>
</tr>
<tr>
<td>QSTINV</td>
<td>Times called 106</td>
<td>106</td>
</tr>
<tr>
<td>QSTCCT</td>
<td>Number of procedures called 114</td>
<td>114</td>
</tr>
<tr>
<td>QSTXCT</td>
<td>Calls to MI complex instr 122</td>
<td>122</td>
</tr>
<tr>
<td>INLCPU</td>
<td>Inline CPU us 130</td>
<td>130</td>
</tr>
<tr>
<td>CUMCPU</td>
<td>Cumulative CPU us 151</td>
<td>151</td>
</tr>
<tr>
<td>INLELP</td>
<td>Inline elapsed us 172</td>
<td>172</td>
</tr>
<tr>
<td>CUMELP</td>
<td>Cumulative elapsed us 193</td>
<td>193</td>
</tr>
<tr>
<td>QIISDR</td>
<td>Inline synchronous DB reads 214</td>
<td>214</td>
</tr>
<tr>
<td>QIISNR</td>
<td>Inline synchronous non-DB reads 218</td>
<td>218</td>
</tr>
<tr>
<td>QIISDW</td>
<td>Inline synchronous DB writes 222</td>
<td>222</td>
</tr>
<tr>
<td>QIISNW</td>
<td>Inline synchronous non-DB writes 226</td>
<td>226</td>
</tr>
<tr>
<td>QIIADR</td>
<td>Inline asynchronous DB reads 230</td>
<td>230</td>
</tr>
<tr>
<td>QIIANR</td>
<td>Inline asynchronous non-DB reads 234</td>
<td>234</td>
</tr>
<tr>
<td>QIIADW</td>
<td>Inline asynchronous DB writes 238</td>
<td>238</td>
</tr>
<tr>
<td>QIIANW</td>
<td>Inline asynchronous non-DB writes 242</td>
<td>242</td>
</tr>
<tr>
<td>QSWI01</td>
<td>Inline software counter 1 246</td>
<td>246</td>
</tr>
<tr>
<td>QSWI02</td>
<td>Inline software counter 2 254</td>
<td>254</td>
</tr>
<tr>
<td>QSWI03</td>
<td>Inline software counter 3 262</td>
<td>262</td>
</tr>
<tr>
<td>QSWI04</td>
<td>Inline software counter 4 270</td>
<td>270</td>
</tr>
<tr>
<td>QIIIPWA</td>
<td>Inline I/O pending waits 278</td>
<td>278</td>
</tr>
<tr>
<td>QIIISWA</td>
<td>Inline synchronous I/O waits 282</td>
<td>282</td>
</tr>
<tr>
<td>QSWC01</td>
<td>Cumulative software counter 1 286</td>
<td>286</td>
</tr>
<tr>
<td>QSWC02</td>
<td>Cumulative software counter 2 294</td>
<td>294</td>
</tr>
<tr>
<td>QSWC03</td>
<td>Cumulative software counter 3 302</td>
<td>302</td>
</tr>
<tr>
<td>QSWC04</td>
<td>Cumulative software counter 4 310</td>
<td>310</td>
</tr>
<tr>
<td>QCISDR</td>
<td>Cumulative synchronous DB reads 318</td>
<td>318</td>
</tr>
<tr>
<td>QCISNR</td>
<td>Cumulative synchronous non-DB reads 322</td>
<td>322</td>
</tr>
<tr>
<td>QCISDW</td>
<td>Cumulative synchronous DB writes 326</td>
<td>326</td>
</tr>
<tr>
<td>QCISNW</td>
<td>Cumulative synchronous non-DB writes 330</td>
<td>330</td>
</tr>
<tr>
<td>QCIAADR</td>
<td>Cumulative asynchronous DB reads 334</td>
<td>334</td>
</tr>
<tr>
<td>QCIAANR</td>
<td>Cumulative asynchronous non-DB reads 338</td>
<td>338</td>
</tr>
<tr>
<td>QCIAADW</td>
<td>Cumulative asynchronous DB writes 342</td>
<td>342</td>
</tr>
<tr>
<td>QCIAANW</td>
<td>Cumulative asynchronous non-DB writes 346</td>
<td>346</td>
</tr>
<tr>
<td>QTSPL</td>
<td>Task pool ID 350</td>
<td>350</td>
</tr>
<tr>
<td>QTSMI</td>
<td>MI task flag Y or N 352</td>
<td>352</td>
</tr>
<tr>
<td>QTSXST</td>
<td>Task existed at start: Y or N 353</td>
<td>353</td>
</tr>
<tr>
<td>QTSXSP</td>
<td>Task existed at stop: Y or N 354</td>
<td>354</td>
</tr>
<tr>
<td>QCPLLNOM</td>
<td>Long MI complex description 355</td>
<td>355</td>
</tr>
<tr>
<td>PGMDSO</td>
<td>Program description 405</td>
<td>405</td>
</tr>
<tr>
<td>QPRPNM</td>
<td>Long procedure name 475</td>
<td>475</td>
</tr>
</tbody>
</table>

Appendix E. Program and MI Complex Instructions summary and Stats’ analysis file descriptions 245
Stats Hierarchical file G_STATSH and G_STATSHP

G_STATSH is the Stats Hierarchical default analysis output file.

G_STATSHP is the Stats Hierarchical Where Used analysis output file.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Field description</th>
<th>Output buffer position</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLLVL</td>
<td>Call Level 1</td>
<td>1</td>
</tr>
<tr>
<td>PCSTS</td>
<td>Partial Count Status 6</td>
<td>6</td>
</tr>
<tr>
<td>LIBNAM</td>
<td>Library Name 7</td>
<td>11</td>
</tr>
<tr>
<td>PGMNAM</td>
<td>Program Name 17</td>
<td>16</td>
</tr>
<tr>
<td>MICPXNM</td>
<td>MI Complex Instruction 27</td>
<td>22</td>
</tr>
<tr>
<td>MODNAM</td>
<td>Module Name 36</td>
<td>31</td>
</tr>
<tr>
<td>PRCNAM</td>
<td>Procedure Name 46</td>
<td>45</td>
</tr>
<tr>
<td>CALLCOUNT</td>
<td>Times Called 302</td>
<td>50</td>
</tr>
<tr>
<td>CALLMADE</td>
<td>Calls Made 313</td>
<td>55</td>
</tr>
<tr>
<td>CALLMICPX</td>
<td>Calls to MI Complex Inst 324</td>
<td>60</td>
</tr>
<tr>
<td>INELPUS</td>
<td>Inline Elapsed us 335</td>
<td>65</td>
</tr>
<tr>
<td>INCPUUS</td>
<td>Inline CPU us 346</td>
<td>70</td>
</tr>
<tr>
<td>INPCPUUS</td>
<td>Inline Percent CPU 357</td>
<td>75</td>
</tr>
<tr>
<td>INCPUPI</td>
<td>Inline CPU us per Call 361</td>
<td>80</td>
</tr>
<tr>
<td>INCOUNT01</td>
<td>Inline Counter 1 373</td>
<td>85</td>
</tr>
<tr>
<td>INPCOUNT01</td>
<td>Inline Percent Counter 1 384</td>
<td>90</td>
</tr>
<tr>
<td>INCOUNT02</td>
<td>Inline Counter 2 388</td>
<td>95</td>
</tr>
<tr>
<td>INPCOUNT02</td>
<td>Inline Percent Counter 2 399</td>
<td>100</td>
</tr>
<tr>
<td>INCOUNT03</td>
<td>Inline Counter 3 403</td>
<td>105</td>
</tr>
<tr>
<td>INPCOUNT03</td>
<td>Inline Percent Counter 3 414</td>
<td>110</td>
</tr>
<tr>
<td>INCOUNT04</td>
<td>Inline Counter 4 418</td>
<td>115</td>
</tr>
<tr>
<td>INPCOUNT04</td>
<td>Inline Percent Counter 4 429</td>
<td>120</td>
</tr>
<tr>
<td>INSDR</td>
<td>Inline Synch DB Read 433</td>
<td>125</td>
</tr>
<tr>
<td>INPSDR</td>
<td>Inline Percent Synch DB Read 444</td>
<td>130</td>
</tr>
<tr>
<td>INSNR</td>
<td>Inline Synch Non-DB Read 448</td>
<td>135</td>
</tr>
<tr>
<td>INPSNR</td>
<td>Inline Percent Synch Non-DB Read 459</td>
<td>140</td>
</tr>
<tr>
<td>INSDW</td>
<td>Inline Synch DB Write 463</td>
<td>145</td>
</tr>
<tr>
<td>INPSDW</td>
<td>Inline Percent Synch DB Write 474</td>
<td>150</td>
</tr>
<tr>
<td>INSNW</td>
<td>Inline Synch Non-DB Write 478</td>
<td>155</td>
</tr>
<tr>
<td>INPSNW</td>
<td>Inline Percent Synch Non-DB Write 489</td>
<td>160</td>
</tr>
<tr>
<td>INSPWA</td>
<td>Inline I/O Pend Waits 493</td>
<td>165</td>
</tr>
<tr>
<td>INPSPW</td>
<td>Inline Percent I/O Pend Wait 504</td>
<td>170</td>
</tr>
<tr>
<td>INSSWA</td>
<td>Inline Synch I/O Waits 508</td>
<td>175</td>
</tr>
<tr>
<td>INPSSWA</td>
<td>Inline Percent Synch I/O Wait 519</td>
<td>180</td>
</tr>
<tr>
<td>INADR</td>
<td>Inline Asynch DB Read 523</td>
<td>185</td>
</tr>
<tr>
<td>INPADR</td>
<td>Inline Percent Asynch DB Read 534</td>
<td>190</td>
</tr>
<tr>
<td>INANR</td>
<td>Inline Asynch Non-DB Read 538</td>
<td>195</td>
</tr>
<tr>
<td>INPANR</td>
<td>Inline Percent Asynch Non-DB Read 549</td>
<td>200</td>
</tr>
<tr>
<td>INADW</td>
<td>Inline Asynch DB Write 553</td>
<td>205</td>
</tr>
<tr>
<td>INPADW</td>
<td>Inline Percent Asynch DB Write 564</td>
<td>210</td>
</tr>
<tr>
<td>INANW</td>
<td>Inline Asynch Non-DB Write 568</td>
<td>215</td>
</tr>
<tr>
<td>INPANW</td>
<td>Inline Percent Asynch Non-DB Write 579</td>
<td>220</td>
</tr>
<tr>
<td>INSD</td>
<td>Inline Synch DB I/O 583</td>
<td>225</td>
</tr>
<tr>
<td>INPSD</td>
<td>Inline Percent Synch DB I/O 594</td>
<td>230</td>
</tr>
<tr>
<td>INSN</td>
<td>Inline Synch Non-DB I/O 598</td>
<td>235</td>
</tr>
<tr>
<td>INPSN</td>
<td>Inline Percent Synch Non-DB I/O 609</td>
<td>240</td>
</tr>
<tr>
<td>INAD</td>
<td>Inline Asynch DB I/O 613</td>
<td>245</td>
</tr>
<tr>
<td>INPAD</td>
<td>Inline Percent Asynch DB I/O 624</td>
<td>250</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>INAN</td>
<td>Inline Asynch Non-DB I/O</td>
<td>628</td>
</tr>
<tr>
<td>INPAN</td>
<td>Inline Percent Asynch Non-DB I/O</td>
<td>639</td>
</tr>
<tr>
<td>INS</td>
<td>Inline Synch I/O</td>
<td>643</td>
</tr>
<tr>
<td>INPS</td>
<td>Inline Percent Synch I/O</td>
<td>654</td>
</tr>
<tr>
<td>INA</td>
<td>Inline Asynch I/O</td>
<td>658</td>
</tr>
<tr>
<td>INPA</td>
<td>Inline Percent Asynch I/O</td>
<td>669</td>
</tr>
<tr>
<td>INW</td>
<td>Inline Waited-on I/O</td>
<td>673</td>
</tr>
<tr>
<td>INPW</td>
<td>Inline Percent Waited-on I/O</td>
<td>684</td>
</tr>
<tr>
<td>CUELPUS</td>
<td>Cumulative Elapsed us</td>
<td>688</td>
</tr>
<tr>
<td>CUCPUS</td>
<td>Cumulative CPU us</td>
<td>699</td>
</tr>
<tr>
<td>CUCOUNT01</td>
<td>Cumulative Counter 1</td>
<td>710</td>
</tr>
<tr>
<td>CUCOUNT02</td>
<td>Cumulative Counter 2</td>
<td>721</td>
</tr>
<tr>
<td>CUCOUNT03</td>
<td>Cumulative Counter 3</td>
<td>732</td>
</tr>
<tr>
<td>CUCOUNT04</td>
<td>Cumulative Counter 4</td>
<td>743</td>
</tr>
<tr>
<td>CUSDR</td>
<td>Cumulative Synch DB Read</td>
<td>754</td>
</tr>
<tr>
<td>CUSNR</td>
<td>Cumulative Synch Non-DB Read</td>
<td>765</td>
</tr>
<tr>
<td>CUSDW</td>
<td>Cumulative Synch DB Write</td>
<td>776</td>
</tr>
<tr>
<td>CUSNW</td>
<td>Cumulative Synch Non-DB Write</td>
<td>787</td>
</tr>
<tr>
<td>CUSPWA</td>
<td>Cumulative I/O Pend Waits</td>
<td>798</td>
</tr>
<tr>
<td>CUSSWA</td>
<td>Cumulative Synch I/O Waits</td>
<td>809</td>
</tr>
<tr>
<td>CUADR</td>
<td>Cumulative Asynch DB Read</td>
<td>820</td>
</tr>
<tr>
<td>CUANR</td>
<td>Cumulative Asynch Non-DB Read</td>
<td>831</td>
</tr>
<tr>
<td>CUADW</td>
<td>Cumulative Asynch DB Write</td>
<td>842</td>
</tr>
<tr>
<td>CUANW</td>
<td>Cumulative Asynch Non-DB Write</td>
<td>853</td>
</tr>
<tr>
<td>CUSD</td>
<td>Cumulative Synch DB I/O</td>
<td>864</td>
</tr>
<tr>
<td>CUSN</td>
<td>Cumulative Synch Non-DB I/O</td>
<td>875</td>
</tr>
<tr>
<td>CUAD</td>
<td>Cumulative Asynch DB I/O</td>
<td>886</td>
</tr>
<tr>
<td>CUAN</td>
<td>Cumulative Asynch Non-DB I/O</td>
<td>897</td>
</tr>
<tr>
<td>CUS</td>
<td>Cumulative Synch I/O</td>
<td>908</td>
</tr>
<tr>
<td>CUA@</td>
<td>Cumulative Asynch I/O</td>
<td>919</td>
</tr>
<tr>
<td>CUW</td>
<td>Cumulative Waited-on I/O</td>
<td>930</td>
</tr>
<tr>
<td>PRCTYP</td>
<td>Procedure Type MI or LIC</td>
<td>941</td>
</tr>
<tr>
<td>REFID</td>
<td>Reference ID for Pgm/Mod/Prc</td>
<td>942</td>
</tr>
<tr>
<td>CLRREFID</td>
<td>Caller Reference ID of Pgm/Mod/Prc</td>
<td>953</td>
</tr>
</tbody>
</table>
Glossary

ASM. Auxiliary Storage Management: A SLIC component that processes all requests to allocate or deallocate disk storage space. Requests to ASM can be counted with PEX Stats and traced and analyzed with PEX ASM Trace and Analysis.

Asynchronous. When applied to disk I/O operations, the requestor is not waiting for a response. The requestor is proceeding without knowing the status of the request. Contrast with Synchronous.

Bracketing. Events that define the beginning and end of a sequence of operations. Usually used to determine the elapsed time or resource usage within that operational sequence.

Call Level. The relative position of a program or procedure in a job's program stack. The highest (or first) level of control is identified as Call Level 0. The next level down, that is, the program called by the Call Level 0 program is Call Level 1. This continues to subsequent levels in sequence. If a job terminates at other than Call Level 0, there is not an orderly, user controlled termination. Instead, the operating system and SLIC roll up the job back through Call Level 0.

CL. Command Language. archaic: Control Language.

CLP. Command Language Program. A i5/OS program written in command language source.

CPP. Command Processing Program. The program called to perform an i5/OS CL Command.

CQE. Classic Query Engine.

Cumulative. The amount of resource used within a program or MI instruction and all the programs or MI instructions that it or its subordinate programs call.

Flat. A PEX Statistics collection mode in which there is one row per unique called program/procedure or MI instruction call per job.

GUI. Graphical User Interface. Often used to mean a Windows menu and mouse actions interface to a program or system function.

Hierarchical. A PEX Statistics collection mode in which there can be multiple rows per program/procedure or MI instruction call per job depending on the calling program name.

HLL. High Level Language: RPG, COBOL, JAVA, or C on the i5.

ILE. Integrated Language Environment®. A program architecture and implementation used on System i to enhance programmer productivity, improve performance, and reduce implementation restrictions of the system's Original Program Model (OPM). There is an IBM Redbooks publication that provides an excellent treatment of the architecture called Moving to Integrated Language Environment for RPG IV, GG24-4358.

Inline. The amount of resource used within a program or MI instruction.

Job. A separate, independent, and asynchronous stream of executable instructions that require time and machine resources to operate.

LIC Assist routine. A sequence of instructions that are part of SLIC that are invoked without going through the MI Instruction routing function. They perform a specific function, but do not have the formal externally defined interface specification requirements that an MI Instruction has. These functions include date conversion, overlapped unaligned data movement, or other functions in which using it as a directly invoked SLIC function is preferable to using generated code in a MI program. Their use is not an external option.

Logical Disk I/O. The movement of a database record into or out of a job's program's work area from/to a system level data base main storage buffer.

MI Complex Instruction. Machine instructions to the System i Machine Interface issued by operating system and application programs to do their normal work. MI complex instructions include functions like finding the pointer to an object (*MATPTR - materialize pointer), writing records sequentially to a file (*INSSDSE - insert sequential data space entries), converting a character value to numeric (*CVTCN,LB - convert character to numeric), creating a duplicate object (*CRTDOBJ - create duplicate object), and so on.

MI Program. A collection of executable instructions bound together in a specific format whose execution observes specific machine architected rules enforced by the machine's SLIC programs.

MI. The Machine Interface. Defines how the MI Programs (i5/OS and applications) must interface with the machine to perform work.
**OPM.** Original Program Model. The first program implementation and architecture on the System i predecessor systems. It was characterized as awkward in its handling of program to program call issues, such as environment scoping and overall performance. It was replaced with ILE.

**PDC.** PEX Data Collection. A SLIC component that provides all PEX associated data collection functions.

**Pending I/O Wait.** A job/task wait on a previously scheduled in-progress I/O operation. The operation can have been scheduled by either the current job/task or some other job/task.

**Service Program.** Relative to PEX Stats data collection, it is a program that needs special consideration during creation to ensure that it is enabled for collection. Its ENBPRFCOL value must be *ENTRYEXIT.*

**SLIC Program.** System Licensed Internal Code program. Part of the base system (not optional).

**SMP.** Symmetric Multi Processing. A i5 processor option that can be used to enhance the throughput rate of SQL, OPNQRYF, Query400, Index Build, Index Maintenance, and file reorganization.

**SQE.** SQL Query Engine.

**Synchronous I/O Wait.** Waiting for a disk I/O request to complete before continuing.

**Synchronous.** A type of disk request in which the issuing job/task must wait for completion before it can continue processing.

**Thread.** A separate and unique flow of control within a job. A thread runs its procedures asynchronously to other threads running within the job. The term thread is an above-TIMI term; the LIC actually uses the term task count.

A single job can have one or many threads. It always has one single primary thread (also called an initial thread) and may have zero to many secondary threads. The work performed by a job is the sum of work performed by all threads within that job plus the work that is done for that job by other system tasks or jobs.

A LIC task does not have threads.

**UDG.** iDoctor’s User Defined Graph.

**UDQ.** iDoctor’s User Defined Query.

**usecs.** Abbreviation for microseconds. Most PEX Stats CPU and Elapsed times are shown in microseconds.

**VRM.** Version, Release, and Modification level

**Widgets.** A generic term used to define your system’s or an application’s unit of work. It could be the number of trucks leaving a back dock, the number of loan applications processed, the number of patients seen, and so on. It is the user’s responsibility to define it; the system’s performance collection facilities do not have application awareness.

For more information about this topic, investigate PEX Analyzer’s User Transaction Trace functions.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

Using the SQL examples in this book

We have purposely shown most of our example SQL statements in this book in Example type figures. If you have Adobe Reader to read the PDF of this book, you can easily copy and paste the SQL example to some text editing tool, such as Microsoft® Word, or to an actual SQL interface, such as Job Watcher’s Data Viewer - SQL Query view interface. You can find the PDF at:

ftp://www.redbooks.ibm.com/redbooks/SG247457

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 251. Note that some of the documents referenced here may be available in softcopy only.

➤ IBM iDoctor iSeries Job Watcher: Advanced Performance Tool, SG24-6474

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services
Index

Numerics
80-20 rule 149
Cumulative percentage 149

A
access code 18, 24
Activation Group 169
Activity Level
job loosing it 192
add a key to the sort sequence 64
ALLOBJ 21
Analysis output
member to data’s job id 101
view to member 101
authority requirements 21
Auxiliary Storage Management (ASM) 166, 190
Discussion 167
file attributes processing 236

Call Level
chart 108
definition 249
discussion 107, 109
Caller Reference Id
definition & discussion 109
Caution
PEX Definition and collection naming 35, 54
CFINTnn 42
Child Analysis 136
output data view 115
Report 137
Running 135
classes
MI Complex instructions 72
Coefficient of Variation
COV 149
Collected job name 103
Collecting the current job 195
collection duration 34, 43
collection modes 8
Collection overhead
duration 43
Job selection 36
Collection Properties 187, 189
collection properties
job level information 62
Collection properties task tab view 42
Collection Scheduling 44
Collections
multiple simultaneous 35
overlapping 35
Component
Flat view SQL 146
Resource usage 144
Concurrent collection
generic selection 90
limitations 90
concurrent PEX data collections 54
Connecting to a system 22
connections list 22
Conventions used in this book
Menu options 3
SQL syntax 3
COV
Coefficient of Variation 149
CPU usage
false positive 84
Create Library prompt 36
Creating New Fields 206
CRTCPRDXDATA
Create PEX Data command 51
CRTCPRDXDTA command 29, 52
Cumulative percent distribution
Inline CPU time 150
Cumulative data
definition 10
Cumulative Elapsed Time
usage 70
Cumulative values
Call Levels 111

D
data base file
Analyzing Opens & Closes 81
close 70, 237
open 70, 75, 237
Data Queues 169
Data Viewer
Save View As 153
decimal digits 207
default PEX Stats Data by Program 58
Deleted records 164
Dynamic Priority Scheduling
job demotion at TSE 192

E
ENBPFFRCOL 74
Create parameter 59
enabled for collection 59
FULL 59
End Collection
menu options 49
Ending a Collection from iDoctor 49
ENDPEX
command discussion 195
mixing modes 54
ENSOBJ
  from RPG FEOD 168
Entering your own SQL SELECT Statement 217
ENTRYEXIT
  ENBPFRCOL parameter 59

event counters
  assignments 189
  hierarchical drill down 149
  specification 178
  types 190
Events count
  Collection 48
  Monitor with ENDPEX 48
  updating with F5 48
examples
  list of PEX Stats analysis example views 15

F
  fastpath
    steps to get started quickly 11
FEOD(N)
  bypasses FEOD ENSOBJ activity 242
  replaces FEOD use in some cases 168
field selection
  Toggle Selected 205
Field Selection and Order 204
Flat Mode 8
Flyover text
  MI Complex Instructions 59
  programs 59
  source library & file QIDRGUI/QAPGMDESCS 59
flyover text 58
FRCRATIO(1)
  causes ENSOBJ activity 242
free trial period expires 19
free trial version 18
FULL
  ENBPFRCOL use 59

G
  General rule
    when to do more analysis 72
Generic job selection 37, 39, 44
Graph definition
  Interface discussion 218
  saving a 224
Group By tab 212

H
  Hier to Flat view
    analysis 122
    creation 117
Hierarchical Mode 9

I
  icon
    iDoctor desktop 21
iDoctor desktop icon 21
iDoctor performance 195
iDoctor’s CPU time slice
  changing 196
iDoctor’s Run Priority
  changing 196
Inline data
  definition 10
Inline Elapsed Time
  definition 69
Instruction Class
  defined 72
IPL
  PEX Data not retained 29

J
  job queue
    PEX Analyzer 101
Job Resource Usage Summary View 82
job selection
  Current job 37
  Limit 38
Job Selection menus 38, 40
Job Watcher
  finding indirect costs 80
  high Cumulative Elapsed 61
  index build 167
  infrastructure comparison 15
  Lock Wait Events 193
  Page Fault information 193
  possible use 6, 61, 63, 69, 81, 83, 86, 169–170, 242
  SLIC program view 9
  wait type breakdown 14
Job/Task Selection 36, 46
JOBQ 24, 55

L
  Library submenu 184
Lic Task
    REFID 109
LR 241

M
  Member name 101
Member Selection 202
MGTCOL
  Object Processing 50
MGTCOL object 29
MI Instructions
  Analysis 72
  markers for investigation 120
  Selected instruction summary 240
  multiple concurrent data analyses 54
Multiple Job Analyses 101
My Connections menu 22

N
  New SQL query
icon 218
non-enabled programs 111

O
Obtaining PEX Analyzer 17–18
OPM programs
enabled for collection 59
ORDER BY 67
Field Selection 68

P
Page Fault
what job/program caused it? 170
Partial Count Status 110
PEX Analyzer/Return trace 59
PEX Definition
PEX Analyzer supplied 28, 46
QAPEXDFN file 28
Renaming STATSFLAT 35
user defined 28, 46
PEX Definition
diDoctor modifies 35
PEX events 190
Counter Assignments 189
Definition menu example 180
IBM Web Site 190
stgevt 190
suggested starter set 180
PEX Filter
example 232
PEX Options 34, 42
PEX PROFILE
High Inline CPU use drill down 63
possible alternative analysis tool 84
PEX Stats Files
G_STATSFJB Flat by Job/Pgm 245
G_STATSFPG Flat by Program 244
G_STATSH Hierarchical 246
PEX Trace
Task Switch 193
TPROF possible alternative analysis tool 84
Program Activation
CALLER 168
High Event Counts 192
NAMED 169
NEW 169
Program full initiation 79
Program name 58
Programs
elapsed runtime 6
PRTPEXRPT
Print PEX Report command 8
PTFs 20

Q
QAPEXDFN file
PEX Definitions file 43
QEZDEBUG 240

QQUERYDEGREE
parameters 171
Query
Record Selection List capabilities 77
Query Definition
interface 202
Saving a 215
working with 216
Query Properties
analysis file member name 103
getting there 67
view 67
quickstart
steps to get started quickly 11

R
Ratio of Cumulative to Inline 83
Record Quick View 138
using 138
record selection 209
Record Selection Filter List 77
Redbooks Web site 251
Contact us xiv
Reference Id 109
REFID
Reference Id 109
relative change
applicability to non-collection numbers 60
before & after 60
Remote Command String 46
Resource Recording
Inline & Cumulative data 112

S
Save View As 153
Saving the view’s data on the PC 153
saving your new query definition 215
scalability 12–13, 163, 167
Scope
Setting the Query value 216
SECADM 21
Service Programs
finding the unenabled ones 197
when programs don’t show up 74
Service programs
identifying enablement setting 59
Signing on to a system 23
SLIC programs 15
SLIC tasks 42
SMP
use with DB2 170
Sort 64
Sort By 210
Sort By tab 210
Sort keys
ALWAYS use Order By 210
properties 64
Specifying Sort Keys 210
SQL
Application and Program Performance Analysis Using PEX Statistics on IBM i5/OS
Application and Program Performance Analysis Using PEX Statistics on IBM i5/OS

Find where to invest to improve job performance

This IBM Redbooks publication is intended for use by those generally familiar with most of the iSeries IBM-provided performance tools available through the i5/OS operating system’s commands and the additional cost Performance Tools for iSeries, 5722-PT1, licensed program.

i5/OS comes with a detailed program level performance data collection capability called the Performance Explorer (PEX). i5/OS commands supporting the collection include Add PEX Definition, Start Performance Explorer, and End Performance Explorer. One of the Performance Explorer (PEX) collection options is called Statistics (*STATS), which collects the program level performance statistics, including CPU usage, disk I/O activity, and the occurrence of certain i5/OS and System i microcode level events. The Print PEX Report function of 5722-PT1 provides a basic view of this *STATS data.

PEX Statistics provides a richer interface for collection and analysis of the *STATS performance data than is available through the i5/OS PEX command and the Print PEX Report output.

Find application "hot spots"

Analyze job program flow