IBM OMEGAMON for IMS on z/OS 5.5.0

Bottleneck Analysis Reference

Note

Before using this information and the product it supports, read the information in "Product legal notices" on page 57.

July 2024 Edition

This edition applies to version 5, release 5, modification 0, of IBM OMEGAMON for IMS on z/OS (product number 5698-T02) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Bottleneck Analysis Collector overview

The Bottleneck Analysis component permits only one collector no matter how many users log on to Bottleneck Analysis. In this way, all users can access the same data.

The attached Bottleneck Analysis collector begins data sampling when you start the OMEGAMON interface. If your site modifies the startup parameters for the Bottleneck Analysis component and the collector is not attached by default, you can attach the collector manually from the OMEGAMON interface, either when the OMEGAMON interface initializes, or at a z/OS console.

When you attach the collector, it initializes itself to receive control commands. The collector does not begin sampling at this time. It waits for further instructions from the Bottleneck Analysis component.

The Bottleneck Analysis component must receive control statements before beginning collection. Either issue **IDEG** and **BEGN** commands in command mode or use the IDEG from the OMEGAMON interface before you issue the **BEGN** command.

Note: Some of the collector control commands can be issued only when the collector is not sampling. The collector stops sampling if it does not receive a **BEGN** command or if it receives an **END** command. An active collector cannot accept these commands because they affect the way the collector allocates the data areas it creates when it receives a **BEGN** command.

Allocation of virtual storage

When the collector begins to sample the processing states of all transactions that flow through IMS, it allocates virtual storage to hold the data that it collects. This area, often referred to as the *counter* or *bucket*, is divided into two parts: the short-term area and the long-term area. The default parameters include an option to allocate space to hold database I/O wait information by individual database name. If you use this option, the collector allocates an extra area, which it also divides into short-term and long-term sections.

The data areas contain counters for overall transaction activity, and also for activity that is broken down by transaction group. The Bottleneck Analysis component supports up to 30 groups. You specify to which groups a transaction type belongs. You can use the transaction groups to target analysis to specific workloads, such as transactions that came from a terminal or that used a PSB. The Bottleneck Analysis component includes 10 groups that correspond to transaction classes 1–10.

Execution states for transactions

When the collector allocates the data areas in virtual storage, it begins observing transactions that flows through your IMS system and records their execution states in its counters.

A *processing state*, also called an *execution state*, is either what a transaction is doing or why it is waiting. The Bottleneck Analysis collector considers each transaction to be in one of the three following execution states:

Non-competing
The collector considers a transaction to be non-competing when it is in a state where IMS cannot process it even if it were the only IMS transaction in an IMS system that runs on a dedicated z/OS® processor. For example, the collector considers a transaction to be non-competing when it is queued for a program that is stopped.

Competing
The collector considers a transaction to be competing when it is in a state where it is eligible to be processed. At any instant, a single transaction is either in a competing or non-competing execution state. There is no overlap.

Competing transactions are either using the processor, doing I/O, or waiting in line behind other transactions for their turn to use the processor and do I/O. Transactions might wait in the message queue, as a transaction does when the only MPP that is servicing its class is busy. Or transactions

might wait in a dependent region, as a transaction does when it is queued for an IMS latch. Wherever the wait occurs, however, a competing transaction either always uses the processor or completes I/O if it is the only transaction in an IMS system on a dedicated processor. z/OS might use the processor or complete I/O on behalf of the transaction. For example, for the resolution of a page fault, work is completed on behalf of the transaction in addition to the loading of a load module overlay segment.

Executing
The collector considers a transaction to be executing when an IMS ITASK processes it. A dependent region might acquire it for processing, that is, the transaction is represented by a Partition Specification Table (PST) in the control region. Transactions that are actively processed by the teleprocessing logic of IMS are also considered to be executing. Such transactions are executing under a Communications Line Block (CLB), which represents a BTAM line or a VTAM®node to IMS instead of a PST.

The collector considers all executing transactions as competing transactions although all competing transactions might not be executing, for example, the transactions in the message queue.

Note: The terms non-competing, competing, and executing can be applied to both the execution states measured by Bottleneck Analysis and to the transactions found by the collector in those states. The topics in this material refer to both usages of these terms.

Transactions that execute in Bottleneck Analysis terms do not necessarily execute in z/OS terms. For example, you can assume that a transaction is running if the application program uses the processor. Bottleneck Analysis also considers a transaction that is waiting in the scheduling process for another transaction to release the block mover to be executing from the time a PST acquires it.

You can display all the transactions in each execution state or you can restrict the Bottleneck Analysis display to show only a single transaction group. The **MDEX** and **PDEX** displays group the execution states. For more information, see “Execution state groups for Bottleneck Analysis transactions” on page 24.

User-defined intervals

At a time interval that you can define, the Bottleneck Analysis collector samples the processing states of all transactions that flow through IMS and records the data it gathers in virtual storage.

At the end of the interval, the collector discards the data that it collects and restarts data collection by zeroing its counters. The collector restarts data collection to maintain current data for users of the realtime monitor. The collector supports two clear intervals, corresponding to its short-term and long-term data areas:

- You set the short-term interval so that the data you see from the short-term counters is from the recent past. The default short-term interval is 5 minutes.
- You set the long-term interval so that the data you see from the long-term counters gives you a longer perspective on the performance of the system that you are studying. The default long-term interval is 30 minutes.

When the long-term interval expires, the Bottleneck Analysis component clears both the long-term and short-term buckets. The collector continues to gather data until you stop it. If more than one display controller is running under the OMEGAMON Classic interface, the collector responds to commands in the order that they are issued. The collector stops gathering data and frees its bucket areas when it receives an **END** command.

Note: Because the Bottleneck Analysis component is a sample-driven collector and the Response Time Analysis component is an event-driven collector, do not directly compare the degradation values to the response time data values.

Chapter 2. Accessing the Bottleneck Analysis component

To access the Bottleneck Analysis component, you can use either the Tivoli® Enterprise Portal or the Bottleneck Analysis menu screens that are available on the OMEGAMON Classic interface (which is also referred to as the OMEGAMON® for IMS Classic interface). You can also use the OMEGAMON enhanced 3270 user interface to display Bottleneck Analysis data. The examples in this material illustrate the Bottleneck Analysis component in the OMEGAMON Classic interface.

Before you begin

To start the Bottleneck Analysis component automatically when you start the OMEGAMON Classic interface, follow the PARMGEN configuration procedures in the *IBM OMEGAMON for IMS on z/OS: Planning and Configuration Guide*.

Note: If your site modifies the startup parameters for the Bottleneck Analysis component and the collector is not attached by default, you can attach and start the collector manually.

Procedure

You access Bottleneck Analysis features from the **Bottleneck Analysis** (KDIBTL) menu panel, which is available by using option B on the **OMEGAMON for IMS Main Menu** panel in the OMEGAMON Classic interface.

1. The **Main Menu** panel, which is shown in the following figure, is the entry into OMEGAMON Classic components.

```
 _________________ ZMENU    VTM     OI-II    V530./C I9IC 01/06/15 16:01:52    B
> Help/News PF1           Exit PF3          Keys PF5          Command Mode PF12
> Return to CUA PA2       Colors PF18
>                 Enter a selection letter on the top line.
================================================================================
>               OMEGAMON for IMS Performance Monitor Main Menu

_ E  EXCEPTIONS ....... Current and potential system problems, latch conflicts
_ R  RESPONSE TIME .... Transaction response times (RTA users)
_ B  BOTTLENECKS ...... Resource contention (bottleneck analysis) (DEXAN users)
_ H  APPLICATION HIST.. Application History

_ M  MONITOR .......... IMS status, graphs, and time controlled operations
_ W  WORKLOAD ......... PSBs, DMBs, transactions, regions, and classes
_ Y  OTMA ............. OTMA status, TMEMBERs, and TPIPEs
_ L  LINES ............ Terminals, nodes, and lines
_ A  ALL POOLS ........ Communication, database, and program pools
_ C  COMPONENTS ....... I/O, logging, storage, and control blocks/modules

_ F  FAST PATH ........ IMS Fast Path information
_ O  OTHER SYSTEMS .... External subsystems (DB2 and MQ) and XRF information

_ T  TOOLS ............ Operator tools
_ P  PROFILE .......... Profile maintenance and session settings
>
================================================================================
```

Figure 1. Main menu

Note: If you are using the Bottleneck Analysis component in a Database Control (DBCTL) environment, your screen might look different from the one in Figure 1 on page 3. Because transactions do not exist in a DBCTL environment, fields that relate to transactions are not displayed. See “DBCTL environment” on page 51 for a list of commands that are not applicable in a DBCTL environment.

2. Use option **B** on the **Main Menu** panel to proceed to the **Bottleneck Analysis** (KDIBTL) menu panel, which is shown in the following figure.

```
 _________________ KDIBTL   VTM     OI-II    V530./C I91C 01/06/15 16:04:11    B

> Help PF1                    Exit PF3
>              Enter a selection letter on the top line.
================================================================================
>                             Bottleneck Analysis

_ A  EXECUTING ..... Factors affecting executing transactions
_ B  COMPETING ..... Wait breakdown of transactions competing for resources
_ C  CONTROL ....... Start/stop the DEXAN collector and control data collection
_ D  OPTIONS ....... Select eligible performance groups and other options

================================================================================
```

*Figure 2. **Bottleneck Analysis** menu*

3. Use the options on the **Bottleneck Analysis** panel to start collecting data, set sampling options, display Bottleneck Analysis information, and perform other tasks that are associated with the Bottleneck Analysis component.

What to do next

If the Bottleneck Analysis collector is not attached by default when the OMEGAMON Classic interface starts, review the following topics for information on how to attach the collector and start data collection.

- "Attaching the Bottleneck Analysis collector" on page 4
- "Starting Bottleneck Analysis data collection" on page 7

Attaching the Bottleneck Analysis collector

You can issue instructions to attach the Bottleneck Analysis collector if the collector is not already attached.

About this task

Typically, you use PARMGEN to configure OMEGAMON Classic to automatically attach and start the Bottleneck Analysis collector when OMEGAMON Classic initializes. If your site does not automatically attach the collector or if the collector was detached, you can manually attach it.

- "Attaching the Bottleneck Analysis collector from the OMEGAMON Classic interface" on page 4
- "Attaching the Bottleneck Analysis collector when OMEGAMON Classic initializes" on page 5
- "Attaching the Bottleneck Analysis collector at a z/OS console" on page 6

Attaching the Bottleneck Analysis collector from the OMEGAMON Classic interface

You can attach the Bottleneck Analysis collector by using the **IDEG** command from the OMEGAMON Classic interface.

About this task

If the Bottleneck Analysis collector is not automatically attached when you start the OMEGAMON Classic address space or it was detached, you can use the **IDEG** command in the OMEGAMON Classic interface to attach it.

Procedure

1. From the **Bottleneck Analysis** (KDIBTL) menu panel, select option **C, CONTROL,** to proceed to the **Start/Stop the DEXAN Collector and Control Data Collection** (KDIBEGN) panel, which is shown in the following figure.

```
 _________________ KDIBEGN  VTM     OI-II    V530./C 191C 04/30/15 13:44:39    B
> Help PF1               Back PF3              Up PF7                Down PF8
================================================================================
>        Start/Stop the DEXAN Collector and Control Data Collection

> To start the collector, enter a hyphen (-) preceding IDEG.            .
 IDEG   >> OI200: Use "-" in column 1 to attach the DEXAN collector   <<

> To begin data collection, remove the > preceding BEGN.
>begn

> To suspend data collection, remove the > preceding END.
>end

> To stop the collector, remove the > preceding DTCH.
>dtch

================================================================================
```

Figure 3. Start/Stop the DEXAN Collector (KDIBEGN) panel

2. Attach the collector by placing a hyphen (**-**) in column 1 before the **IDEG** command, then press Enter. You might see a message that the collector is being attached, as shown in the following figure.

   Note: After you enter **-IDEG**, the **-** in column 1 is replaced by a blank.

```
 _________________ KDIBEGN  VTM     OI-II    V530./C 191C 04/30/15 13:45:39    B
> Help PF1               Back PF3              Up PF7                Down PF8
================================================================================
>        Start/Stop the DEXAN Collector and Control Data Collection

> To start the collector, enter a hyphen (-) preceding IDEG.
 IDEG   >> OI207: DEXAN collector is being attached <<

> To begin data collection, remove the > preceding BEGN.
>begn

> To suspend data collection, remove the > preceding END.
>end

> To stop the collector, remove the > preceding DTCH.
>dtch

================================================================================
```

Figure 4. Attaching the Bottleneck Analysis collector by using IDEG

Note: -IDEG can also be entered in command mode rather than by using the OMEGAMON Classic interface panel.

What to do next

The **-IDEG** command attaches the Bottleneck Analysis collector; however, it does not begin sampling. To begin data sampling, see "Starting Bottleneck Analysis data collection by using the BEGN minor command" on page 7.

Attaching the Bottleneck Analysis collector when OMEGAMON Classic initializes

When OMEGAMON Classic initializes, you can use the **START DEXAN** command to automatically attach the Bottleneck Analysis collector.

About this task

You can use PARMGEN to configure OMEGAMON Classic to automatically attach and start the Bottleneck Analysis collector when OMEGAMON Classic initializes. If your installation does not automatically attach the Bottleneck Analysis collector and you want it automatically attached, use PARMGEN or follow this procedure to change the startup parameters.

Procedure

Member KOIDEX*mp* in *rhilev*.RKANPARU contains the command to attach the Bottleneck Analysis collector. KOIDEX*mp* is created during the configuration process and is run by the KOI*mp*P00 PARM file when you start OMEGAMON for IMS Classic.

1. Enter the following OMEGAMON Classic interface command in member KOIDEX*mp*:

   ```
   START DEXAN
   ```

   Note: START DEXAN automatically attaches the Bottleneck Analysis collector, but it does not start data sampling. To automatically attach the collector and start data sampling when OMEGAMON Classic initializes, add the **IDEG=BEGN** option to the **START DEXAN** command in KOIDEX*mp*:

   ```
   START DEXAN,IDEG=BEGN
   ```

2. Review KOI*mp*P00 in *rhilev*.RKANPARU and ensure the line that contains EXEC KOIDEX*mp* is not commented.

   If there is an asterisk (*) in front of the EXEC statement for KOIDEX*mp*, delete the asterisk.

Results

When you start the OMEGAMON Classic address space, the Bottleneck Analysis collector is automatically attached. If you specified the IDEG=BEGN keyword, then data sampling also starts.

Note: When you start OMEGAMON Classic, the value for the GLOBAL parameter in the startup procedure identifies the KIPGLB member in the RKIPGLBL DD data set that contains the global Bottleneck Analysis defaults, including default transaction groups.

What to do next

To begin data sampling, if it is not automatically started, see "Starting Bottleneck Analysis data collection by using the BEGN minor command" on page 7.

Attaching the Bottleneck Analysis collector at a z/OS console

You can enter the **START DEXAN** command from a z/OS console to attach the Bottleneck Analysis collector.

About this task

If the Bottleneck Analysis collector is not automatically attached when you start the OMEGAMON Classic address space or it was detached, you can enter the **START DEXAN** command from a z/OS console to attach it.

Procedure

At a z/OS console, use the MVS™ MODIFY command to enter **START DEXAN** as shown in the following example.

```
MODIFY mpimsid,START DEXAN
```

where

mp
Two-character MPREFIX set during installation. The MPREFIX must match the MPREFIX in the OMEGAMON Classic startup procedure.

imsid
Four-character IMS subsystem identifier that is specified in the startup procedure.

Note: START DEXAN attaches the Bottleneck Analysis collector, but it does not start data sampling. To attach the collector and start data sampling, use **START DEXAN,IDEG=BEGN** as shown in the following example.

```
MODIFY mpimsid,START DEXAN,IDEG=BEGN
```

Results

The Bottleneck Analysis collector is attached. If you specified the IDEG=BEGN keyword, then data sampling also starts.

What to do next

To begin data sampling, if it is not started, see "Starting Bottleneck Analysis data collection by using the BEGN minor command" on page 7.

Starting Bottleneck Analysis data collection

You can issue instructions to the collector to start Bottleneck Analysis data collection.

About this task

You can enter the command to start data collection in command mode or use the OMEGAMON Classic menu interface. Additionally, if you want data collection to start when OMEGAMON Classic initializes, you can set startup options to automatically begin collection.

- "Starting Bottleneck Analysis data collection by using the BEGN minor command" on page 7
- "Starting Bottleneck Analysis data collection when OMEGAMON Classic initializes" on page 9

Starting Bottleneck Analysis data collection by using the BEGN minor command

Use the **BEGN** command in the OMEGAMON Classic interface to begin data sampling.

Before you begin

Any Bottleneck Analysis minor command, such as **BEGN**, must be preceded by the **IDEG** major command.

About this task

If the Bottleneck Analysis collector is attached, but data collection is not active, you can use the **BEGN** command in the OMEGAMON Classic interface to begin data sampling.

Procedure

You can enter commands in command mode or use the OMEGAMON Classic menu interface. **BEGN** must be preceded by **IDEG**.

- You can enter the **IDEG** and **BEGN** commands by using the following steps.
  a) Enter the **IDEG** major command.
  The results from entering **IDEG** are shown in the following example.

```
 IDEG   >> Collector not active <<
```

  Notice the message that shows that the collector is not active.
  b) Enter the **BEGN** minor command.
  The results from entering **BEGN** are shown in the following example.

```
 IDEG   >> Collector not active <<
>begn   >> Collector now active <<
```

The message for **BEGN** shows that the collector is now active.

Note: After you enter the **BEGN** command, the command comments itself out after it runs by placing a comment character (**>**) in its label field.

c) Optional: Press **Enter** again and **IDEG** shows how long sampling has been active.

```
 IDEG   >> Elapsed time= 33 SEC #samples(short)=7, #samples(long)=71 <<
>begn   >> Collector now active <<
```

IDEG indicates that data sampling started 33 seconds ago.

- You can also use the Bottleneck Analysis panels in the OMEGAMON Classic menu interface to start data sampling.

a) From the **Bottleneck Analysis** (KDIBTL) menu panel, select option **C** to proceed to the **Start/Stop the DEXAN Collector and Control Data Collection** (KDIBEGN) panel, which is shown in the following figure.

The **IDEG** command is automatically entered for you on this panel.

```
 _______________ KDIBEGN  VTM     OI-II    V530./C 191C 05/01/15 13:44:39    B
> Help PF1               Back PF3               Up PF7                Down PF8
================================================================================
>        Start/Stop the DEXAN Collector and Control Data Collection

> To start the collector, enter a hyphen (-) preceding IDEG.            .
 IDEG   >> Collector not active <<

> To begin data collection, remove the > preceding BEGN.
>begn

> To suspend data collection, remove the > preceding END.
>end

> To stop the collector, remove the > preceding DTCH.
>dtch

================================================================================
```

*Figure 5. IDEG on the **Start/Stop the DEXAN Collector** (KDIBEGN) panel*

Notice the **IDEG** message that shows that the collector is not active.

b) Type over the comment character (**>**) that precedes **begn** with a blank, then press **Enter**.

The results are shown in the following figure.

```
 _______________ KDIBEGN  VTM     OI-II    V530./C 191C 05/01/15 13:45:39    B
> Help PF1               Back PF3               Up PF7                Down PF8
================================================================================
>        Start/Stop the DEXAN Collector and Control Data Collection

> To start the collector, enter a hyphen (-) preceding IDEG.            .
 IDEG   >> Collector not active <<

> To begin data collection, remove the > preceding BEGN.
>begn   >> Collector now active <<

> To suspend data collection, remove the > preceding END.
>end

> To stop the collector, remove the > preceding DTCH.
>dtch

================================================================================
```

*Figure 6. BEGN on the **Start/Stop the DEXAN Collector** (KDIBEGN) panel*

The message for **BEGN** shows that the collector is now active.

Note: After you enter the **BEGN** command, the command comments itself out after it runs by placing a comment character (**>**) in its label field.

c) Optional: Press **Enter** again.

The **IDEG** command shows how long sampling has been active.

Note: The other commands on the panel do not run because they are preceded by the comment character.

```
 _________________ KDIBEGN  VTM     OI-II    V530./C 191C 05/01/15 13:46:00    B
> Help PF1               Back PF3               Up PF7                Down PF8
================================================================================
>        Start/Stop the DEXAN Collector and Control Data Collection

> To start the collector, enter a hyphen (-) preceding IDEG.             .
 IDEG   >> Elapsed time=  21 SEC, #samples(short)=42, #samples(long)=42 <<

> To begin data collection, remove the > preceding BEGN.
>begn   >> Collector now active <<

> To suspend data collection, remove the > preceding END.
>end

> To stop the collector, remove the > preceding DTCH.
>dtch

================================================================================
```

*Figure 7. IDEG elapsed time on the **Start/Stop the DEXAN Collector** (KDIBEGN) panel*

IDEG indicates that data sampling started 21 seconds ago.

What to do next

The values in the KIPGLB global definition member that is in effect control the collector. However, you can issue minor commands of **IDEG** to override the defaults, as described in Chapter 3, "Controlling Bottleneck Analysis data collection," on page 15.

Starting Bottleneck Analysis data collection when OMEGAMON Classic initializes

When OMEGAMON Classic initializes, you can use the **IDEG=BEGN** keyword on the **START DEXAN** command to instruct the Bottleneck Analysis collector to automatically begin sampling immediately after the collector is attached.

Before you begin

Review the instructions in "Attaching the Bottleneck Analysis collector when OMEGAMON Classic initializes" on page 5.

About this task

You can use PARMGEN to configure OMEGAMON Classic to automatically attach and start the Bottleneck Analysis collector when OMEGAMON Classic initializes. If your installation attaches the collector, but does not automatically start data sampling, you can follow this procedure to change the startup parameters so that data sampling begins automatically.

Procedure

1. Add the **IDEG=BEGN** option to the **START DEXAN** command in the KOIDEX*mp* member in *rhilev*.RKANPARU.

   ```
   START DEXAN,IDEG=BEGN
   ```

2. Review KOI*mp*P00 in *rhilev*.RKANPARU and ensure the line that contains EXEC KOIDEX*mp* is not commented.

   If there is an asterisk (*) in front of the EXEC statement for KOIDEX*mp*, delete the asterisk.

Results

When you start the OMEGAMON Classic address space, the Bottleneck Analysis collector is automatically attached and data sampling begins.

Note: When you start OMEGAMON Classic, the value for the GLOBAL parameter in the startup procedure identifies the KIPGLB member in the RKIPGLBL DD data set that contains the global Bottleneck Analysis defaults, including default transaction groups.

Completing Bottleneck Analysis data collection

You can issue instructions to the Bottleneck Analysis collector to stop data collection and detach the collector from the OMEGAMON Classic interface.

About this task

You can enter the commands to stop data collection and to detach the collector in command mode or use the OMEGAMON Classic menu interface. You can also detach the collector from a z/OS console.

- "Ending Bottleneck Analysis data collection by using the END minor command" on page 10
- "Detaching the Bottleneck Analysis collector from the OMEGAMON Classic interface" on page 12
- "Detaching the Bottleneck Analysis collector at a z/OS console" on page 13

Ending Bottleneck Analysis data collection by using the END minor command

You can use the **END** minor command in the OMEGAMON Classic interface to instruct the Bottleneck Analysis collector to stop sampling. You might want to use this command to change some of the options, for example, the database I/O reporting option.

Before you begin

Any Bottleneck Analysis minor command, such as **END**, must be preceded by the **IDEG** major command.

Procedure

You can enter commands in command mode or use the OMEGAMON Classic menu interface.

- Enter the **IDEG** and **END** commands in command mode as shown in the following steps.
  a) Enter the **IDEG** major command.
  The results from entering **IDEG** are shown in the following example.

```
 IDEG   >> Elapsed time=10:53 MN, #samples(short)=107, #samples(long)=1305 <<
```

  Notice the message that shows the elapsed time of the current sample.
  b) Enter the **END** minor command.
  The results from entering **END** are shown in the following example.

```
 IDEG   >> Elapsed time=15:48 MN, #samples(short)=96, #samples(long)=1893 <<
>end    >> Collector now dormant <<
```

  The message for **END** shows that the collector is now dormant.

  Note: After you enter the **END** command, the command comments itself out after it runs by placing a comment character (**>**) in its label field.
- You can also use the Bottleneck Analysis panels in the OMEGAMON Classic menu interface to end data sampling.
  a) From the **Bottleneck Analysis** (KDIBTL) menu panel, select option **C** to proceed to the **Start/Stop the DEXAN Collector and Control Data Collection** (KDIBEGN) panel, which is shown in the following figure.
  The **IDEG** command is automatically entered for you on this panel.

```
 _________________ KDIBEGN  VTM     OI-II    V530./C 191C 05/02/15 13:44:39    B
> Help PF1               Back PF3               Up PF7                Down PF8
================================================================================
>        Start/Stop the DEXAN Collector and Control Data Collection

> To start the collector, enter a hyphen (-) preceding IDEG.            .
 IDEG   >> Elapsed time= 5:07 MN, #samples(short)=16, #samples(long)=616 <<

> To begin data collection, remove the > preceding BEGN.
>begn

> To suspend data collection, remove the > preceding END.
>end

> To stop the collector, remove the > preceding DTCH.
>dtch

================================================================================
```

*Figure 8. Active IDEG on the **Start/Stop the DEXAN Collector** (KDIBEGN) panel*

Notice the **IDEG** message that shows the elapsed time of the current sample.

b) Type over the comment character (**>**) that precedes **end** with a blank, then press **Enter**.

The results are shown in the following figure.

```
 _________________ KDIBEGN  VTM     OI-II    V530./C 191C 05/02/15 13:45:57    B
> Help PF1               Back PF3               Up PF7                Down PF8
================================================================================
>        Start/Stop the DEXAN Collector and Control Data Collection

> To start the collector, enter a hyphen (-) preceding IDEG.            .
 IDEG   >> Elapsed time= 5:25 MN, #samples(short)=51, #samples(long)=651 <<

> To begin data collection, remove the > preceding BEGN.
>begn

> To suspend data collection, remove the > preceding END.
>end    >> Collector now dormant <<

> To stop the collector, remove the > preceding DTCH.
>dtch

================================================================================
```

*Figure 9. END on the **Start/Stop the DEXAN Collector** (KDIBEGN) panel*

The message for **END** shows that the collector is now dormant.

Note: After you enter the **END** command, the command comments itself out after it runs by placing a comment character (**>**) in its label field.

Results

The collector remains attached, but dormant, waiting for a **BEGN** command to instruct it to start collecting information again.

What to do next

When the collector is dormant, you can change collection options that cannot be changed when it is active. If you stopped sampling to change those options, change them, then restart data sampling.

Detaching the Bottleneck Analysis collector from the OMEGAMON Classic interface

You can detach the Bottleneck Analysis collector by using the **DTCH** command from the OMEGAMON Classic interface.

Before you begin

Any Bottleneck Analysis minor command, such as **DTCH**, must be preceded by the **IDEG** major command.

Procedure

You can enter commands in command mode or use the OMEGAMON Classic menu interface. You can detach the collector regardless of whether it is active or not. In the following examples, the collector is not active when **DTCH** is entered.

- Enter the **IDEG** and **DTCH** commands in command mode as shown in the following steps.
  a) Enter the **IDEG** major command.

     The results from entering **IDEG** are shown in the following example.

     ```
      IDEG   >> Collector not active <<
     ```

  b) Enter the **DTCH** minor command.

     The results from entering **DTCH** are shown in the following example.

     ```
      IDEG   >> Collector not active <<
     >dtch   >> Collector Detached <<
     ```

     The message for **DTCH** shows that the collector is detached.

     Note: After you enter the **DTCH** command, the command comments itself out after it runs by placing a comment character (**>**) in its label field.

- If you want to enter the **DTCH** command from the Bottleneck Analysis panels in the OMEGAMON Classic menu interface, follow these steps.
  a) From the **Bottleneck Analysis** (KDIBTL) menu panel, select option **C** to proceed to the **Start/Stop the DEXAN Collector and Control Data Collection** (KDIBEGN) panel, which is shown in the following figure.

     The **IDEG** command is automatically entered for you on this panel.

     ```
      _________________ KDIBEGN  VTM     OI-II    V530./C 191C 05/02/15 14:44:39    B
     > Help PF1               Back PF3               Up PF7                Down PF8
     ================================================================================
     >        Start/Stop the DEXAN Collector and Control Data Collection

     > To start the collector, enter a hyphen (-) preceding IDEG.            .
      IDEG   >> Collector not active <<

     > To begin data collection, remove the > preceding BEGN.
     >begn

     > To suspend data collection, remove the > preceding END.
     >end

     > To stop the collector, remove the > preceding DTCH.
     >dtch

     ================================================================================
     ```

     *Figure 10. IDEG with Collector not active message on **KDIBEGN***

  b) Type over the comment character (**>**) that precedes **dtch** with a blank, then press **Enter** to detach the collector.

     The results are shown in the following figure.

```
 _________________ KDIBEGN  VTM     OI-II    V530./C 191C 05/02/15 14:45:57    B
> Help PF1               Back PF3              Up PF7               Down PF8
================================================================================
>        Start/Stop the DEXAN Collector and Control Data Collection

> To start the collector, enter a hyphen (-) preceding IDEG.            .
 IDEG   >> Collector not active <<

> To begin data collection, remove the > preceding BEGN.
>begn

> To suspend data collection, remove the > preceding END.
>end

> To stop the collector, remove the > preceding DTCH.
>dtch   >> Collector Detached <<

================================================================================
```

*Figure 11. DTCH on the **KDIBEGN** panel*

The message for **DTCH** shows that the collector was detached.

Note: After you enter the **DTCH** command, the command comments itself out after it runs by placing a comment character (**>**) in its label field.

Results

The collector is detached. If the collector is active when you enter the **DTCH** command, Bottleneck Analysis stops it, and then detaches it.

What to do next

To attach the collector again, enter the **IDEG** command with a hyphen in column 1, as described in "Attaching the Bottleneck Analysis collector" on page 4. Or enter the **START DEXAN** command at a z/OS console as described in "Attaching the Bottleneck Analysis collector at a z/OS console" on page 6.

Detaching the Bottleneck Analysis collector at a z/OS console

You can detach the Bottleneck Analysis collector by using the **STOP ID=DX** command at a z/OS console.

Procedure

- At a z/OS console, use the MVS MODIFY command to enter **STOP ID=DX** as shown in the following example.

  ```
  MODIFY mpimsid,STOP ID=DX
  ```

  where

  mp
  Two-character MPREFIX set during installation. The MPREFIX must match the MPREFIX in the OMEGAMON Classic startup procedure.

  imsid
  Four-character IMS subsystem identifier that is specified in the startup procedure.
- You can also stop the OMEGAMON Classic address space. If you stop OMEGAMON Classic, all processing ends.
- If you shut down IMS, the entire OMEGAMON Classic address space also ends.

Results

When you enter **STOP ID=DX**, the OMEGAMON Classic address space continues to run. However, the collector is detached. If the collector is active when you enter **STOP ID=DX**, the collector is stopped, then detached.

What to do next

If the OMEGAMON Classic address space is still running you can reattach the collector. To attach the collector again, enter the **IDEG** command with a hyphen in column 1, as described in "Attaching the Bottleneck Analysis collector" on page 4. Or enter the **START DEXAN** command at a z/OS console as described in "Attaching the Bottleneck Analysis collector at a z/OS console" on page 6.

Chapter 3. Controlling Bottleneck Analysis data collection

You can control the Bottleneck Analysis data collection by issuing **IDEG** minor commands.

About this task

You can enter commands in command mode or use the OMEGAMON Classic menu interface.

Excluding BMP transactions from Bottleneck Analysis data collection

You can exclude transactions that are processed by Batch message processing (BMP) programs from your analysis. BMPs are application programs that can perform batch-type processing online and access the IMS message queues for input and output. BMPs can be batch-oriented or transaction-oriented.

Before you begin

- Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.
- You cannot change the **BMPX** option while the EPILOG historical component of the OMEGAMON Classic interface is active. See *IBM OMEGAMON for IMS on z/OS Historical Component (EPILOG) Reference* for information about stopping EPILOG.

Procedure

1. If you want to use command mode, issue one of the **BMPX** minor commands as follows:
   a) Type BMPXON to have the collector ignore transactions that are processed in BMPs. The results from running BMPXON are shown in Figure 12 on page 15.
   b) Type BMPXOFF to have the collector gather data about transactions that are processed in message-driven BMP regions. The results from running BMPXOFF are shown in Figure 12 on page 15.
   c) Type BMPX or BMPX? to display the current setting.

      If you do not set the **BMPX** option, the Bottleneck Analysis component uses the setting for EXCLUDE_MESSAGE_DRIVEN_BMPS in the KIPGLB member that is used by the OMEGAMON Classic address space. The default is OFF.

```
 IDEG   >> Elapsed time=  16 SEC, #samples(short)=35, #samples(long)=35 <<
>bmpxON >> BMP activity is being ignored by DEXAN <<
>bmpxOF >> BMP activity is being recorded by DEXAN <<
```

Figure 12. BMPXOFF and BMPXON commands output

Note: Both BMPXON and BMPXOFF force the collector to clear the long-term and short-term data that is gathered to date, and then place the comment character (>) in the label field so the Bottleneck Analysis component does not re-execute the command.

2. If you want to use the menu interface to specify BMP thread sampling, select option **D, OPTIONS,** from the Bottleneck Analysis menu panel (KDIBTL).
   a) Remove the > character from before the **BMPX** command.
   b) Add **OFF** or **ON** to the **BMPX** command and press Enter.

   Assuming the collector currently ignores message-driven BMPs, when you enter BMPXOFF, the panel updates similar to what is shown in the following figure.

```
 _________________ KDIOPT   VTM     OI-II    V530./C I9IC 04/30/15 13:49:46    B

> Help PF1               Back PF3               Up PF7                Down PF8
================================================================================
>                Eligible Performance Groups and Other Options

> To change the value of an option, enter the new value directly after the
> option.  For the THRS option, enter a space followed by the new value.

 IDEG   >> Elapsed time=   6 SEC, #samples(short)=12, #samples(long)=12 <<
>bmpxOF >> BMP activity is being recorded by DEXAN <<
>clrl   >> both Short and Long term counters have been reset <<
>clrs   >> both short and long counters have been reset <<
 dbsw   >> I/O Analysis will be done by individual database <<
 stim   >> Sample Time Interval= .5 Seconds <<
 thrs   >> 0 is PDEX wait percent threshold <<

>Display contents of all groups:
LSETG99
+ Group #1  (Name=CLASS 1 )
+  Class=1
+ Group #2  (Name=CLASS2  )
+  Class=2
+ Group #3  (Name=GROUP 03)
+  Class=3
+ Group #4  (Name=GROUP 04)
+  Class=4
+ Group #5  (Name=GROUP 05)
+  Class=5
+ Group #6  (Name=GROUP 06)
+  Class=6
+ Group #7  (Name=GROUP 07)
+  Class=7
+ Group #8  (Name=GROUP 08)
+  Class=8
+ Group #9  (Name=GROUP 09)
+  Class=9
+ Group #10  (Name=GROUP 10)
+  Class=10

> To create or add a resource(s) to a group replace the > preceding
> the SETG with a C or an A, replace 'nn' with a group number, and
> replace 'arg' with the required resource definition.
>SETGnn arg
================================================================================
```

Figure 13. Issuing BMPX from the menu interface

Excluding non-message-driven BMPs from Bottleneck Analysis data collection

You can exclude non-message-driven transactions that are processed by Batch message processing (BMP) programs from your analysis. BMPs are application programs that can perform batch-type

processing online and access the IMS message queues for input and output. BMPs can be batch-oriented or transaction-oriented.

Before you begin

- Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.
- You cannot change the **NMSX** option while the EPILOG historical component of the OMEGAMON Classic interface is active. See the *IBM OMEGAMON for IMS on z/OS Historical Component (EPILOG) Reference* for information about stopping EPILOG.

Procedure

- Type NMSXON to have the collector ignore non-message-driven BMPs. The results from running NMSXON are shown in Figure 14 on page 17.
- Type NMSXOFF to have the collector gather data about non-message-driven BMP regions. The results from running NMSXOFF are shown in Figure 14 on page 17.
- Type NMSX or NMSX? to display the current setting.

  If you do not set the **NMSX** option, the Bottleneck Analysis component uses the setting for EXCLUDE_NON_MESSAGE_DRIVEN_BMPS in the KIPGLB member that is used by the OMEGAMON Classic address space. The default is ON.

```
 IDEG   >> Elapsed time=  16 SEC, #samples(short)=35, #samples(long)=35 <<
>nmsxON >> Non-message driven BMP activity is being ignored by DEXAN <<
>nmsxOF >> Non-message driven BMP activity is being recorded by DEXAN <<
```

Figure 14. NMSXOFF and NMSXON commands output

Note: Both NMSXON and NMSXOFF force the collector to clear the long-term and short-term data that is gathered to date, and then place the comment character (>) in the label field so the Bottleneck Analysis component does not re-execute the command.

The **NMSX** command is a functional subset of **BMPX**. If **BMPXON** is in effect, all BMP activity is ignored regardless of the **NMSX** switch, as shown in Table 1 on page 17.

Table 1. NMSX on and off conditions

	NMSXON	**NMSXOFF**
BMPXON	Non-message BMP activity ignored	Non-message BMP activity ignored
BMPXOFF	Non-message BMP activity ignored	Non-message BMP activity is recorded

Setting the long-term clear interval for Bottleneck Analysis data collection

You can use the **CLRL** command to set the interval for clearing the long-term data counters to *nnn* minutes. The long-term interval is set so that the data you see from the long-term counters gives you a longer perspective on the performance of the system.

Before you begin

Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.

Procedure

- Type CLRL with a numerical argument, for example CLRL30, to set the interval for clearing long-term data counters. The results from entering CLRL30 are shown in Figure 15 on page 18.

The Bottleneck Analysis component accepts a value within the range of 1 and 480 minutes (8 hours). A reasonable value for the long-term clear interval is 30 minutes (CLRL30). The value that you enter for **CLRL** must be longer than the short-term clear interval set by **CLRS**.

- Type CLRL to clear both the short-term and long-term counters without changing the clear interval as shown in Figure 15 on page 18.
- Type CLRL? to display the existing long-term clear interval.

  If you do not provide a value for **CLRL**, the Bottleneck Analysis component uses the setting for LONG_TERM_BUCKETS_MINUTES in the KIPGLB member that is used by the OMEGAMON Classic address space. The default is 30.

```
 IDEG   >> Elapsed time=  16 SEC, #samples(short)=35, #samples(long)=35 <<
>clrl  30   >> long-interval counters will be reset every 30 minutes <<
>CLRL   >> both Short and Long term counters have been reset <<
```

Figure 15. CLRLnnn and CLRL commands output

Note: When you change the value of **CLRL**, the collector clears both the short-term and long-term counters, and the comment character (>) is placed in the label field of the **CLRL** command so that the Bottleneck Analysis component does not re-execute it.

Setting the short-term clear interval

You can use the **CLRS** command to set the interval for clearing the short-term data counters to *nn* minutes. The short-term interval must be set so that the data you see from the short-term counters is from the recent past.

Before you begin

Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.

Procedure

- Type CLRS with a numerical argument, for example CLRS5 to set the interval for clearing short-term data counters. The results from entering CLRS5 are shown in Figure 16 on page 18.

  The Bottleneck Analysis component accepts a value within the range 01 - 99. The value that you enter for **CLRS** must be shorter than the long-term clear interval set with **CLRL**. A reasonable value for the short-term clear interval is 5 minutes (CLRS05). Use at least 30 samples for statistical significance; how long it takes to get them depends on the setting of the cycle time.
- Type CLRS to clear both the short-term and long-term counters without changing the clear interval as shown in Figure 16 on page 18.
- Type CLRS? to display the existing short-term clear interval.

  If you do not provide a value for **CLRS**, the Bottleneck Analysis component uses the setting for SHORT_TERM_BUCKETS_MINUTES in the KIPGLB member that is used by the OMEGAMON Classic address space. The default is 5.

```
 IDEG   >> Elapsed time=  32 SEC, #samples(short)=63, #samples(long)=63 <<
>clrs  5   >> short-interval counters will be reset every 5 minutes <<
>CLRS   >> both short and long counters have been reset <<
```

Figure 16. CLRSnn and CLRS commands output

Note: When you change the value of **CLRS**, the Bottleneck Analysis component clears both the short-term and long-term counters, and **CLRS** places the comment character (>) in its label field so that Bottleneck Analysis does not re-execute it.

Performing Bottleneck Analysis data sampling by database name

You can use the **DBSW** minor command to instruct the Bottleneck Analysis collector to allocate virtual storage space to store database I/O wait information by individual database name.

Before you begin

- Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.
- The database sampling option cannot be changed while the Bottleneck Analysis collector is active. If you attempt to do so, the Bottleneck Analysis component displays an error message.

About this task

Allocating and storing virtual storage space by individual database name is required because the Bottleneck Analysis component must allocate counters for each database for each transaction group you define. The Bottleneck Analysis component requires an increase in virtual storage when the product of the number of transaction groups and databases is large.

The number of bytes that the Bottleneck Analysis component requires is as follows:

```
8 * (<number of databases> * (<MAXG>+1))
```

MAXG is the maximum number of transaction groups for which the Bottleneck Analysis component allocated space. For more information about the MAXG command, see the *IBM OMEGAMON for IMS Realtime Commands Reference*.

Procedure

- Type DBSWON to perform I/O analysis by individual database. The results from running DBSWON are shown in Figure 17 on page 19.
- Type DBSWOFF to switch off I/O analysis by individual database. The results from running DBSWOFF are shown in Figure 17 on page 19.
- Type DBSW or DBSW? to display the current database selection option.

```
 IDEG   >> Collector not active <<
>END    >> Collector now dormant <<
>dbswOF >> I/O Analysis will not be done by individual database <<
>dbswON >> I/O Analysis will be done by individual database <<
 IDEG   >> Collector not active <<
>BEGN   >> Collector now active <<
```

Figure 17. Changing the Database Sampling Option (DBSW)

Attention: If the Bottleneck Analysis collector is active, you must stop the collector before you enter DBSWON or DBSWOFF. To stop the collector, issue the **IDEG** major command and the **END** minor command as shown in Figure 17 on page 19.

Note: When you enter DBSWON or DBSWOFF, the **DBSW** command comments itself out after execution by setting its label field to >.

What to do next

After you change the database sampling option, issue the **IDEG** and **BEGN** commands to restart the collector. Figure 17 on page 19 shows this series of commands.

Performing Bottleneck Analysis data sampling by transaction type

You can use the **DOPT** minor command to select which type of transaction you want to analyze. You can control whether the **MDEX** and **PDEX** commands display information about executing, competing, or all transactions including non-competing transactions.

Before you begin

Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.

About this task

The **DOPT** command prevents the Bottleneck Analysis component from washing out the degradation data when the message queue includes many different types of transactions. The **DOPT** command affects only the displays. The Bottleneck Analysis collector continues to gather data on both competing and non-competing transactions. Executing transactions are transactions that are associated with an IMS dependent region.

Procedure

- Type **DOPT** with one of the following arguments to specify the options for gathering transaction data.
  - COMP

    Display information about competing transactions only
  - EXEC

    Display information about executing transactions only
  - TOTAL

    Display information about all transactions
  - ALL

    Display information about all transactions (alias of TOTAL)

  For example, type DOPT EXEC to display information about executing transactions only or type DOPT COMP to display information about competing transactions only. The following figure shows the results from running DOPT EXEC and DOPT COMP.

```
 IDEG   >> Elapsed time= 6:35 MN, #samples(short)=193, #samples(long)=791 <<
>dopt  EXEC  >> Only Executing Transactions Will Be Analyzed <<
>dopt  COMP  >> Only "Competing" transactions will be analyzed <<
```

  Figure 18. DOPT commands output

  Note: When you set the value of **DOPT**, the **DOPT** command comments itself out after execution by setting its label field to >.
- Type DOPT or DOPT? to display the current setting.

  Note: If you do not set the **DOPT** option, the Bottleneck Analysis component uses the setting for DISPLAY_OPTIONS in the KIPGLB member that is used by the OMEGAMON Classic address space. The default is COMP.

Performing Bottleneck Analysis data sampling by time interval

You can use the **STIM** command to control the sample time interval that the Bottleneck Analysis collector uses when you analyze the IMS environment.

Before you begin

Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.

About this task

The Bottleneck Analysis collector analyzes the IMS environment at a fixed sample rate. The **STIM** command controls the sample time interval that the Bottleneck Analysis collector uses.

Procedure

- Type STIM with a numeric argument, STIM*nn*, to set the sample interval.

  The Bottleneck Analysis collector clears both the short-term and long-term data areas before it begins to sample at the new rate, as shown in the following example. You can specify a value from 1 to 99 which the **STIM** command interprets as tenths of a second, to vary the rate from a low of 0.1 seconds to a high of 9.9 seconds. For example, STIM15 indicates a sample time of 1.5 seconds. Use a **STIM** value that is small enough to collect at least 30 samples during the short-term interval so that your data is statistically significant.

- Type STIM, without a numeric argument, to have the Bottleneck Analysis collector display the current sample time.

  If you do not specify a value for **STIM**, the Bottleneck Analysis component uses the setting for SAMPLE_TIME_INTERVAL in the KIPGLB member that is used by the OMEGAMON Classic address space. The default value is 5, which is 0.5 seconds.

```
 IDEG   >> Elapsed time=  32 SEC, #samples(short)=63, #samples(long)=63 <<
>stim5  >> Sample Time Interval= .5 Seconds <<
```

Figure 19. STIM command output

Note: If you change the sample interval, the **STIM** command comments itself out (by placing a **>** in its label field) after it runs.

Suspending the Bottleneck Analysis collector

You can use the **SUSP** command to instruct the Bottleneck Analysis collector to stop sampling temporarily, but not to clear its short-term and long-term buckets. The **SUSP** command freezes the collector data long enough for you to analyze the information by using the **MDEX** or **PDEX** command.

Before you begin

Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.

Procedure

- Issue the **SUSP** minor command of the **IDEG** major command, as shown in Figure 20 on page 21.

```
 IDEG   >> Elapsed time= 3:15 MN, #samples(short)=352, #samples(long)=352 <<
>susp   >> Collector now suspended <<
```

Figure 20. SUSP command output

What to do next

When you finish examining the data, you can start the collector again by using the **RESM** command.

Resuming Bottleneck Analysis collector data sampling

You can use the **RESM** command to instruct the Bottleneck Analysis collector to resume its sampling.

Before you begin

- Use the **RESM** command only after you use the **SUSP** command to suspend the collector. The **RESM** command does not clear any counters, and the data that displays is not representative of the interval during which collection was suspended.
- Any Bottleneck Analysis minor command must be preceded by the **IDEG** major command.

Procedure

- Issue the **RESM** minor command of the **IDEG** major command, as shown in Figure 21 on page 22.

```
 IDEG   >> Elapsed time= 3:15 MN, #samples(short)=352, #samples(long)=352 <<
>resm   >> Collector has been resumed <<
```

Figure 21. RESM command output

Chapter 4. Bottleneck Analysis displays (MDEX/PDEX)

You can view the transaction data that runs on your Information Management system (IMS) system using the **MDEX** and **PDEX** commands.

The **MDEX** and **PDEX** commands control the display of data. Both commands accept arguments that you can use to include all transactions in the display or restrict the display to a single transaction group.

MDEX
Displays average counts of transactions in each of the execution states that the Bottleneck Analysis collector monitors.

PDEX
Displays the percent of time an average transaction spends in each of the execution states that the Bottleneck Analysis collector monitors.

Factors that influence MDEX/PDEX displays

A set of factors determine the data that the **MDEX** and **PDEX** commands display.

- Whether the collector gathered data about message-driven BMPs. The **BMPX** command controls the collection of this data. "Excluding BMP transactions from Bottleneck Analysis data collection" on page 15 describes the **BMPX** command.
- Whether the collector gathered data about transaction DL/I I/O for each individual database, or just for database I/O as a whole. The **DBSW** command controls the collection of this data. "Performing Bottleneck Analysis data sampling by database name" on page 19 describes the **DBSW** command.
- Whether the **MDEX** or **PDEX** command restricts its display to a certain class of execution states. To indicate the class of execution states, put one of a set of characters in the label field (column 1) of the command as "Setting Bottleneck Analysis display commands (MDEX/PDEX)" on page 41 describes.
- Whether the **MDEX** or **PDEX** command restricts its display to a single transaction group. A numeric argument from 01 to 30 follows the command to indicate that **MDEX** or **PDEX** displays data only for transactions in a user-defined group. For example, **PDEX02** causes the Bottleneck Analysis component to display data only for transactions that the collector determined to fall in Group 2. The *IBM OMEGAMON for IMS Realtime Commands Reference* explains how to define groups. The display can also be limited to a single group by using the GRP= keyword to specify the transaction number or name of the group.
- Whether you specify that the display is restricted only to executing transactions, only to competing transactions, or that the display is not restricted. These restrictions prevent unimportant execution states from "washing out" the important ones on a **PDEX** display, or from misleading you on an **MDEX** display. You specify the restrictions by using the **DOPT** minor command as explained in "Performing Bottleneck Analysis data sampling by transaction type" on page 20.
- For **MDEX**, the current display threshold and scaling factor. Only execution states whose short- and long-term average counts are less than the value specified by using the **MTHR** command display. "Setting the display threshold for the MDEX command" on page 44 explains how to vary the display threshold. By setting a scaling factor, the **SCAL** command further controls the display. With it, you can display execution states whose short- and long-term average counts might ordinarily be too small to see. How you specify the scaling factor is explained in "Setting the scaling factor for the MDEX command" on page 44.
- For **PDEX**, the current display threshold. **PDEX** displays only the execution states that account for more than the percentage of an average execution time that is specified by using the **THRS** minor command. If you specified a percentage greater than zero by using **THRS**, the execution state numbers might not add up to 100%. Conversely, setting the percentage greater than zero reduces the size of the display. "Setting the display threshold for the PDEX command" on page 45 describes the **THRS** command.

- For **MDEX** and **PDEX** displays, the volume of activity on your system. **MDEX** and **PDEX** does not display an execution state if there are no transactions. It is impractical to do otherwise because there are so many possible execution states in IMS.
- Whether you are using the Bottleneck Analysis component in a DBCTL environment. Because transactions do not exist in a DBCTL environment, fields on **MDEX** and **PDEX** displays relating to transactions are not displayed. Also, on **MDEX** displays, only a subset of z/OS and IMS wait reasons are displayed.

Execution state groups for Bottleneck Analysis transactions

A processing state, also called an execution state, represents either what a transaction is doing or why it is waiting. Execution states can be logically grouped.

The **MDEX** and **PDEX** commands display the execution states by group. It is important that you understand the concepts of executing, competing, and non-competing transactions.

- CPU Usage
- IMS Scheduling Waits
- Database I/O Waits
- z/OS Waits
- IMS Waits
- Output Waits
- External Subsystem Waits

CPU usage group

Bottleneck Analysis focuses on processor usage. Removing bottlenecks enables transactions to use more CPU per unit of time and, therefore, complete faster.

The individual execution states in the CPU usage group describe how much processor the transactions use and where, for example, in DL/I versus CPU used in the application program. These states are all executing states; they are experienced only by transactions that are running in a dependent region.

Note: Because of their importance, Bottleneck Analysis always displays these execution states, even when you use a label field character to restrict the display to a certain class of execution groups as described in "Setting Bottleneck Analysis display commands (MDEX/PDEX)" on page 41.

You can use the Using CPU states to infer activity that Bottleneck Analysis cannot measure directly. For example, a shift away from Using CPU towards Database I/O, in the absence of increased I/O contention, indicates that the databases that are accessed might require reorganization. An abrupt, massive increase in Using CPU across all transaction types indicates hardware CPU degradation.

Using CPU in APPL (Executing)

The collector includes a transaction in the Using CPU in APPL state when it finds that the application program that is processing the transaction is actually running instructions on a processor. The program had no DL/I function in progress, so the collector ascribes its use of the processor to application processing. Note, however, that the collector counts CPU used by z/OS components such as program fetch in this category

In a normal IMS system, this number is small, even tiny, compared to the other execution states. However, this number is one of the most important statistics you can watch because of the following factors:

- Almost every tuning trick increases this number and decreases the total of the other execution states by the same amount. Indeed, a transaction that executes at 100% in this state is not degraded at all in the Bottleneck Analysis sense of the term unless it is looping.
- Almost every performance problem decreases this number and increases the total of the other execution states by the same amount. For example, if I/O contention slows down access to a

database, the execution profile of the application shifts towards Database I/O and away from this execution state.

Using CPU in IMS (Executing)
The collector found that the application program that is processing the transaction is actually executing instructions on a processor.

The program had a DL/I call in progress, so the collector ascribed its use of the CPU to IMS processing. An example of such processing is the CPU used by DL/I action modules to search a buffer pool for a record before you issue an I/O to retrieve it.

IMS scheduling waits group

When IMS schedules transactions, appropriate resources must be available, otherwise the transaction must wait. There are multiple points within scheduling where waiting can occur. Bottleneck Analysis shows the dynamic characteristics of transactions that are waiting during the scheduling process.

IMS transactions are serviced by application programs that are running within message processing regions (MPPs) or message-driven batch-message processing regions (BMPs). Each region can service one transaction at a time. When more transactions arrive than the active regions can service, IMS holds them in the message queues until the regions complete current work, or until IMS activates more regions.

IMS incurs overhead when it activates an application program within a message region. IMS applications can process multiple transactions each time they are scheduled. The **PROCLIM** parameter of the **TRANSACT** macro limits the number of transactions that IMS services each time the application is activated.

You can customize IMS applications so that transactions are serviced by only one region, or by multiple regions in parallel. The **MAXRGN** parameter of the **TRANSACT** macro controls the number of parallel regions. If the number of transactions queued exceeds the PARLIM, IMS allows another region to service them.

When a transaction is waiting for IMS to schedule it into a region, Bottleneck Analysis distinguishes between the following states:

Block Loader (Executing)
IMS cannot schedule a transaction into a dependent region until its PSB and all the DMBs it requires are loaded from ACBLIB.

When the collector sees IMS routines that are scheduling a transaction while loading the blocks from ACBLIB, it counts that transaction in this category. For more information about this execution state, see the sections about the block mover, PSB pool, and DMB pool execution states.

Block Mover Wait (Executing)
IMS cannot schedule a transaction into a dependent region until IMS loads its PSB and all the DMBs it requires from ACBLIB.

IMS allows only one PST to load blocks at a time. Any others that want to load blocks are put on a queue. This is reasonable because ACBLIB is on a single volume that can have only one I/O on it at a time. Serialization is also a result of the SMGT latch. The transaction that is counted here has a PST attempting to schedule it which the collector found on that queue.

DDIR Block Latch (Executing)
The collector found the transaction that is waiting for the DDIR block latch.

DDIR Pool Latch (Executing)
The collector found the transaction that is waiting for the DDIR Pool latch.

DMB Block Latch (Executing)
The collector found the transaction that is waiting for the DMB block latch.

DMB Pool (Competing)
A transaction cannot be run in a dependent region until the DMBs for all of the databases it uses are in the DMB pool. For those DMBs that are not RESIDENT, IMS must find room for a copy in the pool before scheduling can complete. If space cannot be found, the PST that is trying to schedule

the transaction Iwaits for it. The transaction that is counted here is found waiting for space in the DMB pool when a PST attempted to schedule it. As a result, the PST is not available for use by other transactions.

Note: Space is freed up in the DMB pool by deleting DMB blocks that are not currently in use. Before a DMB block can be deleted, all of the data sets it uses must be closed. The next time that DMB is needed, it must be reloaded and its data sets reopened. OPEN and CLOSE operations halt IMS processing and cause z/OS services to start, and are therefore considered highly undesirable.

Intent Conflict (Competing)
The collector found the transaction that is waiting in the message queue.

An MPP attempted to schedule it, but was unable to because the PSB associated with the transaction indicated an intent towards one of the databases it accesses which the PSB could not honor at the moment, for example, PROCOPT=E when other scheduled PSBs are using the database. Intent conflicts also result from applications that is requesting parallel scheduling when load balancing is disabled at the transaction level.

PDIR Block Latch (Executing)
The collector found the transaction that is waiting for the PDIR block latch.

PDIR Pool Latch (Executing)
The collector found the transaction that is waiting for the PDIR Pool latch.

PSB Block Latch (Executing)
The collector found the transaction that is waiting for the PSB block latch.

PSB Pool (Competing)
The collector found the transaction that is waiting in the message queue. IMS cannot schedule a transaction into a dependent region until its PSB is in the PSB pool.

If the PSB is not RESIDENT or even if it is parallel-schedulable, IMS must find room for a copy of the PSB in the pool before scheduling can proceed. If IMS cannot find space, the scheduling attempt fails and the PST remains idle until an application terminates. Scheduling then proceeds according to the scheduled option for the failing transaction. The transaction that is counted here is found waiting for space in the PSB pool when a PST attempted to schedule it. As a result, the PST is not available for use by other transactions.

PSBW Pool (Competing)
The collector found the transaction that is waiting in the message queue.

There is at least one message processing region available to service this transaction, but scheduling fails because there is not enough available space in the PSB work pool for this transaction to run. The transaction needs space in the PSB work pool for an I/O area, an area for segment search arguments (SSA), and a copy of the user parameter list, for example.

TM Allocate PSB Latch (Executing)
The collector found the transaction that is waiting for the TM allocate PSB latch.

TM Scheduling Latch (Executing)
The collector found the transaction that is waiting for the TM scheduling latch.

Unschedulable (Non-Competing)
The collector found the transaction in the message queue and determined that it could not currently be scheduled, even if there are no other transactions that is blocking it. IMS cannot schedule the transaction for one of the following reasons:

- The transaction type is STOPped.
- The transaction type is USTOPped.
- The class of the transaction type is STOPped.
- The program (PSB) which processes the transaction type is STOPped.
- The program (PSB) which processes the transaction type is not found on ACBLIB.
- One or more databases that are required by the PSB, which processes the transaction type are unavailable, for example, stopped or are not found.

- For transactions destined for an MPP, there are no MPPs running, which are sensitive to the class of the transaction.
- For transactions destined for an MPP, the current scheduling priority is zero.

The collector can count both MPP and BMP transaction types in this state.

Wait for BMP (Non-Competing)
The collector found the transaction that is waiting in the message queue. The transaction is destined for a message-driven BMP region. The BMP is not currently running but without reason, for example an unavailable database. The user submits a job to start BMPs. The IMS does not automatically schedule BMP transactions like it schedules MPPs.

This state is a non-competing state and a situation similar to a batch job that is TYPRUN=HOLD. Nothing can be done by the IMS or those trying to tune it to make such transactions run faster. They cannot run at all until the user chooses to submit the BMP job.

Wait for IFP (Competing)
This state includes the number of transactions queued waiting to be serviced by a fast path application program that is running in a fast path message region. The application is servicing other transactions.

Wait for GU (Competing)
This number indicates the number of transactions queued, which will be serviced by the application that is running in one or more regions. The Wait for GU value is the sum of the remaining **PROCLIM** counts of active regions that currently scheduled that transaction, but it does not exceed the number of queued transactions.

Wait for MPP (Competing)
This number is the number of transactions queued that will not be serviced by an active application for one of the following reasons:

- The application might not be active because message processing regions (MPPs) are currently servicing other transactions.
- The number of transactions that are queued exceeds the cumulative remaining **PROCLIM** values of the regions that are currently processing this transaction. The value of **PARLIM** does allow the application to be scheduled in another message processing region, if one is available.

If you start another MPP with the appropriate class, these transactions can be processed.

Wait for Reschedule (Competing)
This number is the number of transactions queued that will not be serviced by an active application for the following reason:

The number of transactions queued exceeds the cumulative remaining **PROCLIM** values of the regions currently processing this transaction. The value of **PARLIM** does not allow the application to be scheduled in another message processing region, even if one is available.

You must reschedule the application program before these transactions are processed.

Database I/O waits

Many database DL/I requests can be satisfied from records that are already in a buffer pool. Others, however, require that IMS perform physical I/O.

MDEX and **PDEX** show a subtotal for database I/O, and if you specified that such statistics be kept by the collector, it shows the breakdown of the I/O by database. No further breakdown, such as by data set group or individual data set, is available from Bottleneck Analysis.

The individual execution states within this category are the DMB names of each database to which a transaction did I/O.

A wait reason of INDEXPCB indicates I/O activity that was caused by secondary index maintenance. When maintenance is performed on a secondary index, IMS constructs a PCB to that index. Although you might not be processing in secondary sequence, I/Os to a secondary index might be caused by:

- Designing your database such that the segment or fields that are used as the key in secondary sequence are updated heavily by other applications.
- Changes to the primary database. Even if the application program is not using the secondary index, maintenance must be performed on the index, and a PCB constructed for it, when the primary database is changed.

Also, there is no way to tell from just this statistic which stages of I/O processing, for example, active on the UCB or waiting in the logical channel queue, that the I/O operations for the databases are in. Bottleneck Analysis can provide this information.

In addition to IMS/VS full function databases, physical I/O operations that are performed to data entry databases (DEDBs) are reported. If the statistics by database is requested, the area name is shown as the execution state.

DEDB updates are performed by output threads asynchronously after the transaction completes and the changes are logged. I/O operations that are performed by output threads are not a source of degradation for IMS/VS transactions and therefore they are not reflected in the database I/O waits.

z/OS waits

While executing in a message processing region or a batch message processing region, a transaction might be blocked by contention for resources managed by z/OS, as opposed to resources managed by IMS. An example of such a resource is the CPU.

The following individual execution states in this group show you when and where transactions are slowed down by resources managed by z/OS:

Application I/O (Executing)
The collector finds that the transaction is scheduled into an MPP and determines that this MPP issued an I/O request that did not go through DL/I. The collector also noticed that the I/O is not of the type used by program fetch.

Usually, installation standards do not allow applications to perform I/O of the type this section discusses. If a debugging feature that uses an output file remains on when the program is put into production, the standard can be violated.

BLDL I/O (Executing)
When IMS schedules a transaction into a message processing region, its application program must be in virtual storage before it can begin to execute. If the application is not in the PreLoad state, IMS causes the z/OS program fetch to bring it into virtual storage from PGMLIB.

The collector considers a transaction to be in this execution state if it observes that the PDS directory I/O is performed on behalf of a BLDL request, which occurs if IMS does not find the BLDL entry for the module to load in the BLDL look-aside buffer, or if z/OS does not use the one that it finds because PGMLIB is authorized.

CPU Wait (CTL) (Executing)
The collector finds that the transaction is ready to run on a CPU within the IMS control region, but all online CPUs in the complex were running some other work. The CPUs might be running other IMS control region tasks, IMS dependent regions, or memories that are not related to IMS, such as TSO.

Note: The collector does not count itself in this analysis. If the only reason a transaction is not executing on a CPU is that the collector is executing, the collector counts the transaction as Using CPU. This is a standard sampling technique for measuring CPU usage by workload.

Also, the collector is unable to see CPU Wait that results from the execution of work with a higher dispatching priority than itself. For example, the collector cannot see CPU Waits resulting from disabled, global SRB, higher priority address spaces such as *MASTER*, and local SRBs in the IMS CTL region. If there is much of this higher priority activity, the measurements are skewed. Some amount of CPU Wait, and even other states such as Database I/O Wait, is seen as Using CPU.

CPU Wait (DEP) (Executing)
The collector finds that the transaction is ready to run on a CPU within its dependent region, but all online CPUs in the complex are running some other z/OS address space. The CPUs might be running the IMS control region, other dependent regions, or address spaces not related to IMS such as TSO.

Cross Memory Page (Executing)
The collector finds that the transaction is active inside an MPP. The IMS system is running with LSO=X. The application program that is processing the transaction issues a DL/I call which is processed by IMS modules running in the control region, but under the ASCB/TCB of the dependent region through the z/OS Cross Memory Services. DL/I processing suffers a page fault for a page in the private area of the control region.

In this situation, the execution in the dependent region is suspended while the page fault is resolved. Processing by the IMS control task is unaffected.

CSA Page (Executing)
The collector finds that the transaction is active inside an MPP region. However, the transaction cannot run for one of two reasons:

- The application program (or, more likely, some IMS module that is running as a subroutine of the application program) is waiting for a page fault in the common service area (CSA) to be resolved.
- The application program that is processing the transaction is in the ISWITCHed state in the IMS control region. While in the IMS control region, the collector finds that its ITASK is in one of the following states:
  - The ITASK is waiting for a page fault in CSA to be resolved
  - The ITASK is waiting to use the CPU after another ITASK, which took a page fault in CSA and serialized the CTL Task

In either case, the page fault occurs when the available frame queue (AFQ) is empty. The request to resolve the page fault is added to the GFA queue where the collector finds it.

CSA Page (GFA) (Executing)
The collector finds that the transaction is active inside an MPP with its execution blocked for one of two reasons:

- The application program (or, more likely, some IMS module running as a subroutine of the application program) is waiting for the resolution of a page fault for a page in the common service area (CSA).
- The application program that is processing the transaction is in the ISWITCHed state in the IMS control region. While there, the collector finds that its ITASK is waiting for the resolution of a page fault it took in CSA, or waiting to use the CPU behind another ITASK that serializes the CTL Task by taking a page fault in CSA.

In either case, the page fault occurred at a time when the available frame queue (AFQ) is empty, so the request for the resolution of the page fault must be put on the GFA queue, and that is where the collector finds it.

CSA Page (MSDB) (Executing)
The collector finds that the transaction is active inside an MPP or IFP region with its execution blocked. The application program is waiting for the resolution of a page fault for a page in the common service area (CSA) occupied by a pageable main storage database (MSDB).

- The application program (or, more likely, some IMS module running as a subroutine of the application program) is waiting for the resolution of a page fault for a page in the common service area (CSA).
- The application program that is processing the transaction is in the ISWITCHed state in the IMS control region. While there, the collector finds that its ITASK is waiting for the resolution of a page fault in CSA, or waiting to use the CPU behind another ITASK that serializes the CTL task by taking a page fault in CSA.

CTL EXT Priv Page (Executing)
This execution state is the same as the CTL Private Page execution state, except that the private page z/OS referenced is in the extended private area of the control region, which is located above the 16 megabyte addressing line.

CTL Private Page (Executing)
The collector finds that the transaction is active inside an MPP. The application program that is processing the transaction is in the ISWITCHed state in the IMS control region and is found waiting for one of the following reasons:

- Its ITASK suffered a page fault for a page in the private area of the address space of the control region.
- Its ITASK is ready to use the CPU, but it is waiting on the IMS dispatcher's READY queue while the control task is serialized by a page fault taken in the private area by some other ITASK.

The IMS control region can tolerate a small amount of paging because when one ITASK page faults in the CTL region, all ITASKs running in the CTL region halt.

CTL Private Page (GFA) (Executing)
The collector finds that the transaction is active inside an MPP. The application program that is processing the transaction is in the ISWITCHed state in the IMS control region and one of two things happened:

- Its ITASK suffers a page fault for a page in the private area of the address space of the control region.
- Its ITASK is ready to use the CPU, but it is waiting on the IMS dispatcher's READY queue while the control task is serialized by a page fault taken in the private area by some other ITASK.

In either case, the page fault occurred at a time when the Available Frame Queue (AFQ) is empty. As a result, the request for the resolution of the page fault must be put on a special Real Storage Manager (RSM) queue called the General Frame Allocation (GFA) queue while the System Resources Manager (SRM) required more frames to replenish the AFQ.

GFA wait is not a major problem because SRM STEAL is not the only source of replenishment for the AFQ. Every time an address space is physically swapped out, the frames it occupied go on the AFQ. Also, when a FREEMAIN is issued, all frames backing the virtual storage released are made available.

DEP Private Page (Executing)
The collector finds that the transaction is active inside an MPP. The application program that is processing the transaction (or possibly some IMS module running as a subroutine of the application program) is waiting for the resolution of a page fault for a page within its private area.

DEP EXT Priv Page (Executing)
This execution state is the same as the DEP Private Page execution state, except that the private page z/OS referenced is located in the dependent region's extended private area, above the 16 megabyte addressing line.

DEP Private Page (GFA) (Executing)
The collector finds that the transaction is active inside an MPP. The application program that is processing the transaction, or possibly some IMS module running as a subroutine of the application program, is waiting for the resolution of a page fault for a page within its private area. The request for resolution of the page fault is found on the GFA queue, indicating that it occurs when RSM is out of frames (AFQ of 0).

ECSA Page (Executing)
The collector finds that the transaction is active inside an MPP region or an IFP region. However, the transaction cannot run for one of two reasons:

- The application program (or, more likely, some IMS module that is running as a subroutine of the application program) is waiting for a page fault in the extended common service area (ECSA) to be resolved.

- The application program that is processing the transaction is in the ISWITCHed state in the IMS control region. While in the IMS control region, the collector finds that its ITASK is in one of the following states:
  - The ITASK is waiting for a page fault in ECSA to be resolved
  - The ITASK is waiting to use the CPU after another ITASK, which took a page fault in CSA and serialized the CTL Task

In either case, the page fault occurs when the available frame queue (AFQ) is empty. The request to resolve the page fault is added to the GFA queue where the collector finds it.

ECSA Page (MSDB) (Executing)
The collector finds that the transaction is active inside an MPP or an IFP with its execution blocked. The application program is waiting for the resolution of a page fault for a page in the extended common service area (ECSA) occupied by a pageable main storage database (MSDB).

- The application program or, more likely, some IMS module running as a subroutine of the application program, is waiting for the resolution of a page fault for a page in the ECSA.
- The application program that is processing the transaction is in the ISWITCHed state in the IMS control region. While there, the collector finds that its ITASK is waiting for the resolution of a page fault in extended CSA, or waiting to use the CPU behind another ITASK that serializes the CTL task by taking a page fault in extended CSA.

ELPA Page (Executing)
The collector finds that the transaction is active inside an MPP with its execution blocked for one of two reasons:

- The application program, most likely compiler runtime routines copied to LPA or some IMS module running as a subroutine of the application program, is waiting for the resolution of a page fault for a page in the extended pageable link pack area (EPLPA).
- The application program that is processing the transaction is in the ISWITCHed state in the IMS control region. While there, the collector finds that its ITASK is waiting for the resolution of a page fault in extended LPA, or waiting to use the CPU behind another ITASK that serializes the CTL task by taking a page fault in extended LPA.

LPA Page (Executing)
The collector finds that the transaction is active inside an MPP with its execution blocked for one of two reasons:

- The application program, most likely compiler runtime routines copied to LPA or some IMS module running as a subroutine of the application program, is waiting for the resolution of a page fault for a page in the pageable link pack area (PLPA).
- The application program that is processing the transaction is ISWITCHed to the IMS control region. While there, the collector finds that its ITASK is waiting for the resolution of a page fault in LPA, or waiting to use the CPU behind another ITASK which serializes the CTL task by taking a page fault in LPA.

LPA Page (GFA) (Executing)
The collector finds that the transaction is active inside an MPP with its execution blocked for one of two reasons:

- The application program or, more likely, some IMS module running as a subroutine of the application program, is waiting for the resolution of a page fault for a page in the pageable link pack area (PLPA).
- The application program that is processing the transaction is in the ISWITCHed state in the IMS control region. While there, the collector finds that its ITASK is waiting for the resolution of a page fault it took in LPA, or waiting to use the CPU behind another ITASK that serializes the CTL task by taking a page fault in LPA.

In either case, the page fault occurred at a time when the AFQ is empty, so the request for the resolution of the page fault must be put on the GFA queue, and that is where the collector finds it.

Program Fetch I/O (Executing)
When IMS schedules a transaction into a message processing region, its application program must be in virtual storage before it can begin to execute. If the application is not in the PRELOADed state, IMScauses the z/OS program fetch to bring it into virtual storage from PGMLIB.

The collector considers a transaction to be in this execution state if it observes I/O done by PCI fetch. If the module to be loaded is large, program fetch I/O can be time-consuming.

Swapping In (Executing)
The collector finds the transaction that is active inside an MPP region is being swapped in by z/OS. Normally, IMS makes all its dependent regions non-swappable, so you can expect some installation modification.

X-MEM Page (GFA) (Executing)
The collector finds that the transaction is active inside an MPP. The IMS system is running with LSO=X or LSO=S. The application program that is processing the transaction issues a DL/I call that is processed by IMS modules that are running in the control region or DLISAS, but under the ASCB/TCB of the dependent region that uses z/OS Cross Memory Services.

DL/I processing suffers a page fault for a page in the private area of the control region. The collector finds that the request for resolution of the page fault is on the GFA queue, indicating that it occurred when the AFQ is empty. In this situation, execution in the dependent region is suspended while the page fault is resolved. Processing by the IMS control task is unaffected.

IMS waits

While executing in a message processing region or a batch message processing region, a transaction might be blocked by contention for resources managed by IMS as opposed to resources managed by z/OS. An example of such a resource is an IMS latch. The following individual execution states in this group show you when and where transactions are slowed down by resources that are managed by IMS:

ACTL Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the ACTL, that is, the statistics logging latch.

IMS uses the ACTL latch to serialize access to the DC Monitor log buffers and control blocks. When the DC Monitor is active, the control region and all parallel DL/I tasks compete for this latch.

ADSC Directory Latch (Executing)
The collector found that the application program that is running the transaction is waiting for the ADSC directory latch.

The ADSC directory latch is used to serialize access to the ADSC directory. This wait is built on the dynamic control block (CBTS) latch. The ADSC directory latch is obtained in the open and close data entry database (DEOB) modules, and fast path command processing modules.

AUTH Latch (Executing)
The collector found that the application program that is running the transaction is waiting for the AUTH latch. The AUTH latch is the authorized processing latch.

CBTS Latch (Executing)
The collector found that the application program that is running the transaction is waiting for the CBTS latch. IMS uses the CBTS latch to serialize alterations to dynamic control block chains (IPAGES) within dynamic storage management.

Conversation Checkpoint Latch (Executing)
The collector found that the transaction is waiting for the Conversation Checkpoint Latch.

DB System Checkpoint Latch (Executing)
The collector finds the transaction is waiting for the DB System Checkpoint Latch.

DBBP Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the DBBP latch. The DBBP latch serializes accesses to database buffers, and their associated control blocks.

DBCTL Latch (Executing)
The collector finds that a DBCTL thread is waiting for the DBCTL latch. The DBCTL latch is used when a DBCTL thread starts.

DBLK Latch (Executing)
The collector finds that the dependent region is waiting for the DBLK latch during signoff processing, that is, the termination of the region.

DC System Checkpoint Latch (Executing)
The collector finds that the transaction is waiting for the DC System Checkpoint Latch.

DC Terminal Latch (Executing)
The collector finds that the transaction is waiting for the DC Terminal Latch.

DC User Latch (Executing)
The collector finds that the transaction or thread is waiting for the DC User Latch.

DEDB Area Lock (Executing)
The collector finds that the application program that is running the transaction is waiting for the DEDB area lock. The DEDB area lock serializes updates to a DEDB area data set. There is a different lock for each area. The DEDB area lock is obtained in sync point processing.

A DEDB area lock is always taken before the DMAC latch to avoid the possibility of a deadlock.

DEDB Segment (Executing)
The collector finds that the application program that is running the transaction is waiting to obtain control of a resource in a data entry database (DEDB). The resource under control is a unit of work (UOW), and it is represented by an XCRB control block.

DMAC Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the DMAC latch.

The DMAC latch is used to serialize updates to DEDB area data sets. Sync point processing modules require the DMAC latch. Updates are recorded in the log records. The physical update of the DEDB area does not take place until the updates are written to the IMS log.

DMAC Share Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the DMAC share latch. The DMAC share latch is used to serialize access to DMAC control blocks. Open and close area data set processing modules require the DMAC share latch in exclusive mode.

When a fast path message processing region terminates, diagnostic information is copied form the control region to the dependent region. This function requires the DMAC share latch in share mode. Waits occur only for this function because there can be only one requestor of the DMAC share latch in exclusive mode at one time, and these modules are serialized by the ADSC latch.

Fast Path Buffer (Executing)
The collector finds that the application program that is running the transaction is waiting for a fast path buffer. Buffer allocation for an IMS Fast Path Region (IFP) is less than the maximum number of allowable buffers specified by the normal buffer allocation (NBA) execution parameter.

Fast Path IWAIT in Term (Executing)
The collector finds that the fast path application program that is processing the transaction is in the IWAITing state while the application program is terminating.

Fast Path Other Abend (Executing)
The collector finds that the fast path application program that is processing the transaction is in the IWAITing state and the application program is terminating due to an abend or canceled dependent region.

Fast Path Overflow Buffer Allocation (Executing)
The collector finds the application program that is running the transaction is waiting for a fast path overflow buffer.

The IFP reaches the maximum number of buffers allowed as specified by the NBA execution parameter), and overflow processing is in progress. Overflow processing increased the buffer allocation by the amount allowed for overflow (OBA).

Fast Path Overflow Buffer Lock (Executing)
The collector finds that the application program that is running the transaction is waiting for the fast path overflow buffer lock.

If an IFP attempts to allocate buffers beyond the normal number of buffers allowed (NBA), overflow processing takes place. Overflow processing attempts to increase the buffer allocation by the amount allowed for overflow (OBA). The overflow lock is required to allocate more buffers than the number specified by normal buffer allocation. The overflow lock is enqueued by IFP and held until sync point processing, at which time the buffer allocation is returned to normal.

Fast Path Pseudo Abend (Executing)
The collector finds that the fast path application program that is processing the transaction is in the IWAITing state while the transaction processing in the message region is pseudo abended.

Fast Path Resource Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the fast path resource latch.

The fast path resource latch serializes access to the exclusive control resource blocks (XCRBs). XCRBs represent the status of a currently owned and possibly exclusively-controlled resource within a DEDB area.

Fast Path Syncpoint Processing (Executing)
The collector finds the transaction in sync point processing. During this state, the dependent region is using CPU. However, the amount of time spent in this state is small.

Fast Path Sync Lock (Executing)
The collector finds the application program that is running the transactions is waiting for the fast path sync lock.

The fast path sync lock serializes the sync point processing in IFPs with check-point processing and other activities that stop a DEDB area, such as after you issue a /STOP AREA, /STOP ADS, or /DBR AREA command, or if a physical I/O error is detected. The sync point processing function requests the fast path sync lock in share mode. All other activities require the lock to be held in exclusive mode.

FNCB Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the FNCB latch.

The FNCB latch serializes access to the notify control block chain. The notify facility is used for communication by various functions such as open or close a DEDB, and IMS commands.

IRLM Conflict Wait (Executing)
The collector finds that the application program is executing inside a dependent region.

The dependent region issued a request to the IRLM address space and is waiting for a response. The wait can occur when there is a database conflict or when IRLM is waiting for internal IRLM processing.

ISWITCHed to CTL (Executing)
The collector finds that the application program that is running the transaction executed an ISWITCH macro. The Partition Specification Table (PST) is still executing, but in the control region instead of in the dependent region.

This wait reason occurs whenever a PST needs the services of the control region. When this happens, the PST gets ISWITCHed to CTL and waits to request the service that it needs from the control region TCB. The wait is longer if the TCB itself is waiting for control of the CPU, for example, if the control region has a high paging rate.

This occurs under the following conditions:

- I/O to ISAM files
- Obtaining input messages including IMS message queue I/O if it is required

- ISRTing output messages including IMS message queue I/O if it is required
- EOV processing on all database data sets when such processing is required to free up a buffer
- Users issuing large numbers of IMS commands
- Servicing Fast Path Wait For Input (WFI) GU calls
- Completing sync point processing

ISWITCHed to Fast Path TCB (Executing)
The collector finds that the application program that is running the transaction executed an ISWITCH macro. The PST is still executing, but in the control region under the Fast Path TCB.

ISWITCHed to LSO (Executing)
The collector finds that the application program that is running the transaction executed an ISWITCH macro.

The PST is still executing, but in the control region instead of in the dependent region. The ISWITCH is performed by using the local storage option (LSO). Instead of running in the control task, the PST is:

- For LSO=Y, running under a subtask in the control region that is reserved for its use only. LSO=Y costs a substantial amount of CPU, but saves virtual storage in CSA at the cost of virtual storage in the private area of the IMS control region.
- For LSO=X, running under the ASCB/TCB of the dependent region, but in the virtual storage associated with the control region. This occurs for the same reason as LSO=Y, but costs less CPU because of the greater efficiency of z/OS Cross Memory Services over the old WAIT/POST logic.

IWAIT in IMS Dispatcher (Executing)
IWAIT in IMS dispatcher is a measure of the IMS dispatching queue for dependent region activities.

The collector finds that the application program is processing the transaction IWAITing in the IMS dispatcher and cannot ascribe the condition to any of the DL/I oriented states previously listed. As the workload rises, this value also rises. This number is small. If it is not, inform IBM® Software Support.

IWAIT in Term (Executing)
The collector finds that the application program is processing the transaction IWAITing.

It also finds an indication that the application program is terminating. The collector could not attribute the status of the transaction to any of the states previously listed.

Note: This execution state is small. If it is not, inform IBM Software Support. Termination IWAITs might be attributed to sync point processing of an update transaction.

LGMSG I/O (Executing)
The collector finds that the transaction is waiting for I/O on the long message (LGMSG) data set.

LOGL Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the LOGL latch.

A latch is the IMS version of an z/OS suspend lock, which efficiently serializes access to some resource that cannot be used by more than one transaction at a time. The IMS logical logger uses the LOGL latch to serialize access to the buffers that the physical logger writes out to the online log data set (OLDS). If the physical logger gets behind, ITASKs queue on this latch.

LQB Pool Latch (Executing)
The collector finds that the transaction is waiting for the LQB Pool Latch.

LU 6.2 Manager Latch (Executing)
The collector finds that the transaction is waiting for the LU 6.2 Manager (LUM) Latch.

MFS Load (Executing)
The collector finds an I/O in progress against the message format services (MFS) data set. This means that an MFS format block is being loaded so that an input or output message could be formatted by MFS.

MSDB Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the main storage database (MSDB) latch. Whenever an MSDB call or sync processing routine needs to get or release control of an MSDB resource, the MSDB latch must be obtained.

MSDB Segment (Executing)
The collector finds that the application program that is running the transaction is waiting to obtain control of a resource in a main storage database (MSDB). The resource under control is an MSDB segment.

Open/Close Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the open/close latch. The open/close latch serializes resources to fast path databases.

OSAM Buffer Wait
The collector finds a DL/I address space.

Other Abend (Executing)
The collector finds that the application program that is processing the transaction is IWAITing. It also finds an indication that the application program is terminating due to an abend or because the dependent region is canceled.

Other DL/I IWAIT (Executing)
The collector finds the application program that is processing the transaction is in DL/I, but it cannot attribute the condition to any of the DL/I oriented states previously listed. Currently PSTs waiting for space in both the ISAM/OSAM and VSAM pools have the transactions that they are processing counted in this category.

Note: This execution state is small. If it is not, inform IBM Software Support.

Other Latch (Executing)
The collector finds that the application program that is running the transaction is in ISERWAIT (latch wait), but not waiting for any of the latches described previously.

Other Wait (Executing)
The collector finds a transaction is waiting, but cannot ascribe its execution state to any of the categories documented previously. If this execution state appears large, report it to IBM Software Support.

PI Wait (Executing)
The collector finds that the application program that is processing the transaction is IWAITing for a program isolation (PI) resource. Program isolation is the mechanism that allows databases to be simultaneously accessed and even updated by different users.

A PI resource is essentially a DMB number to identify the physical database, and an relative byte address (RBA) to identify the physical block that is held. When two programs attempt to access the same PI resource at the same time, one program must wait.

Pseudo Abend (Executing)
The collector finds that the application program that is processing the transaction is IWAITing and terminating due to a pseudo abend.

QBLKS I/O (Executing)
The collector finds that the transaction is waiting for I/O to complete to the queue blocks data set.

QBUF Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the QBUF latch. The QBUF latch serializes accesses to message queue buffers, and their associated control information.

Queue Manager Latch (Executing)
The collector finds the transaction is waiting for the Queue Manager Latch.

SHMSG I/O (Executing)
The collector finds that the transaction is waiting for I/O on the short message (SHMSG) data set.

SMB Queue Latch (Executing)
The collector finds that the transaction is waiting for the SMB Queue Latch.

SMGT Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the storage management (SMGT) latch. The IMS storage management module (DFSISMN0) uses this latch to serialize access to various control blocks and buffers, such as the message queue, CIOP, CWAP, DMB, and PSB pools.

Sync Point Wait (Executing)
The collector finds that the transaction is waiting for I/O to complete as the result of synchronization (SYNC) point processing.

TCT Block Latch (Executing)
The collector finds that the transaction is waiting for the TCT Block Latch.

TM Subqueue Latch (Executing)
The collector finds that the transaction is waiting for the TM Subqueue Latch.

VSAM Buffer Handler (Executing)
The dependent region of the application is in the VSAM buffer handler.

VTCB Pool Latch (Executing)
The collector finds that the transaction is waiting for the VTCB Pool Latch.

Wait HSSP PVT Pool (Executing)
The collector finds that the application program is executing inside a dependent HSSP BMP region. The application program is waiting for space in the HSSP private pool.

XCNQ Latch (Executing)
The collector finds that the application program that is running the transaction is waiting for the XCNQ latch.

Program isolation uses the XCNQ latch to serialize access to the PI ENQ/DEQ queues and control blocks.

Output waits

When outbound processing of a message is delayed because IMS is waiting for an output-related service or resource, Bottleneck Analysis detects an output wait.

Message on Fast Path Output Queue (Non-competing)
The collector finds that an output message that is generated by the transaction in the output queue is waiting to be dequeued. The fast path output router, which executes as an IMS task in the control region, dequeues fast path output messages.

Message on Non-Fast Path Output Queue (Non-competing)
The collector finds that an output message that is generated by a transaction in the non-fast path output queue is waiting to be dequeued.

External subsystem waits

External subsystem (ESS) waits are those waits in which the collector finds a dependent region that is waiting for a reply from an external subsystem, for example, DB2.

Abort (Rollback) Wait (Executing)
The collector finds that the dependent region is aborting changes.

Commit (Phase 2) Wait (Executing)
The collector finds that the application program is committing changes (Phase 2).

Create Thread Wait (Executing)
The collector finds the dependent region while creating a thread. A thread is the DB2 structure that describes a connection between DB2 and IMS regions.

When a create thread is in progress, DB2 is allocating an application plan to process the SQL CALL. The creation of a DB2 thread might use considerable resources. Associated with create thread processing include plan load, authorization checking, database open, and lock authorization. An application plan defines the DB2 resources that are accessed from an application program.

DB2 Call Wait (Executing)
The collector finds that the application program is running a DB2 or MQ command.

Echo (Check ESS) Wait (Executing)
The collector finds that the application program is processing an echo call.

ESS Other Wait (Executing)
The collector finds that the external subsystem is completing some other action that is not described in this topic.

Ident IMS to ESS Wait (Executing)
The collector finds that the application program is processing an IDENTIFY request to a DB2 subsystem.

Init IMS Attach Wait (Executing)
The collector finds that the dependent region is initializing the connection to DB2.

Prepare to Commit Wait (Executing)
The collector finds that the dependent region is preparing to commit changes to DB2 data.

Resolve Indoubt Wait (Executing)
The collector finds that the dependent region is resolving an indoubt thread.

SQL Call Wait (Executing)
The collector finds the application program while running an SQL call. An IMS application program issues SQL calls to request service from DB2.

Subsys Terminate Wait (Executing)
The collector finds that the dependent region is ending the external subsystem connection.

Sys Not Operating Wait (Executing)
The collector finds that the dependent region is unable to locate the DB2 system.

Term Indent to ESS Wait (Executing)
The collector finds that the dependent region is ending the IDENTIFY to the external subsystem (DB2).

Terminate Thread Wait (Executing)
The collector finds the dependent region while ending a thread.

When the sync point completes, the thread terminates and the plan that is used by the SQL CALL is deallocated unless the region is a WFI. DB2 database close might also occur at terminate thread.

User Sign on ESS Wait (Executing)
The collector finds that the dependent region is processing a SIGNON to DB2 (external subsystem).

User Sign off ESS Wait (Executing)
The collector finds that the dependent region is signing off from the connection to the external subsystem.

Scenario: why restrict MDEX/PDEX displays to exclude non-competing transactions?

Whether you display all transactions or just a single group, you can restrict the Bottleneck Analysis display to executing transactions, competing transactions that includes but is not limited to executing transactions, or include all transactions, both competing and non-competing.

To understand why it is important to exclude non-competing transactions from a display, consider a typical IMS system that queues transactions for BMPs. All day long MPPs issue transactions that request a BMP to update a database. IMS does not actually run the BMP until 3:00 AM when its use of the database does not interfere with the performance of the online inquiry transactions. By 3:00 AM thousands of transactions are in the queue for the BMP. What does a **PDEX** display of this system look like?

If the Bottleneck Analysis component did not exclude such transactions, Wait For BMP might be 100% for the system as a whole. The Bottleneck Analysis component observes other execution states, but it rounds them down to zero in the face of the thousands of BMP transactions. There are a number of ways to remedy this problem:

- You can set the **BMPX** option to ON, which causes the collector to ignore all transactions that are destined for BMPs. The problem with this method is that you cannot track the performance of scheduled BMP regions with the Bottleneck Analysis component.
- You can use the transaction group facility to put all the BMP transactions into group 01 and all others into group 02. Then, you can use PDEX02 to see just MPP traffic. The disadvantage of using this method is that you must update the Bottleneck Analysis group tables each time you add a transaction. Also, you cannot use this mechanism to separate transactions destined for scheduled BMPs (which are worth watching) from transactions that are destined for unscheduled ones (which are not worth watching).

  For more information about the transaction group facility, see the *IBM OMEGAMON for IMS Realtime Commands Reference*.
- You can set the **DOPT** option to COMP, excluding non-competing execution states and the transactions found in them from the displays that **MDEX** and **PDEX** generate. Because Wait For BMP is non-competing, this method solves the problem. This solution has many advantages. The collector still gathers data about BMPs and you can look at it any time. You do not need to define any transaction groups. When a BMP is started, its transaction processing automatically becomes visible. A similar process goes on as messages accumulate for a transaction code with a normal priority of zero and a nonzero limit priority. Until the threshold is reached, the transactions are non-competing, but as soon as enough messages accumulate to increase the priority, all the messages start competing.

You must also limit the displays that are produced by **MDEX** and **PDEX** to executing execution states. What happens to an IMS system when the performance (throughput) of its MPPs degrades because of excessive paging? If the arrival rate does not go down (as it does in a purely conversational environment), the message queues start to build. When there are many messages in the queue, all **PDEX** visibility into the real problem is rounded down to zero. Wait For MPP is 100%. All other states, including Wait For Page, disappear. What can be done in this situation?

If you set the **DOPT** option to EXEC, **MDEX** and **PDEX** ignore Wait For MPP and all other message queue that relates to execution states. The real problem, Wait For Page, is then obvious.

The concepts of competing and executing transaction execution states are significant, but it is a mistake to think that you can always ignore transactions that are not executing, or at least those states that are non-competing. For example, if an MPP results in an abend, it goes out of service and all the transactions queued for it becomes non-competing because they cannot be processed.

Should such transactions be excluded from the displays that are produced by the Bottleneck Analysis component? If the analyst knows about the abend, the answer is probably yes. There is no need to see those transactions. They might wash out performance problems with other transaction types. Conversely, if the analyst does not know that the application abended and is trying to use **PDEX** to find out why the users of the transactions are not being serviced, then the answer is certainly no.

If Bottleneck Analysis exception analysis is available, there is no problem. An exception message informs the analyst of the application program that is going out of service, so the **DOPT** option can be set to COMP without fear of missing important information.

The concepts of competing, executing, and non-competing transactions are as applicable to output messages on the queue as they are to transactions in the input queue. In the complex IMS, z/OS, VTAM environment it is not always possible for the Bottleneck Analysis component to know which state is applicable, but the concepts always are.

A simple example is an IMS system that supports multiple terminals on a BSC line under BTAM. Output messages in the queue for an LTERM that is stopped are non-competing because IMS cannot send the messages even if the messages are the only work in the system.

Messages in the output queue that IMS cannot send now because the line is busy with the output of another message, are competing but not executing. An output message that is actually undergoing output processing by the CLB, which owns the line (its bytes might actually be going down the line, or perhaps the message is being prepared for transmission by MFS) is both competing and executing.

Note: It is important that you understand the concepts of executing, competing, and non-competing transactions.

For a discussion of transaction groups and how to define them, see the *IBM OMEGAMON for IMS Realtime Commands Reference*.

Chapter 5. Customizing and viewing the MDEX and PDEX displays

You can monitor the output from the Bottleneck Analysis component and customize the displays so that the data is specific to your Information Management System (IMS) environment.

About this task

You can configure Bottleneck Analysis displays from the OMEGAMON Classic interface or in command mode.

Setting Bottleneck Analysis display commands (MDEX/PDEX)

You can use the **MDEX** and **PDEX** commands to display transactions that the Bottleneck Analysis collector monitors.

Before you begin

Any Bottleneck Analysis minor command must be preceded by the IDEG major command.

Procedure

- Type the **MDEX** or **PDEX** command as shown in Figure 22 on page 41,

  where *a* represents command label field, *GRP* represents group name or number, and *nn* represents command argument field (DEXAN group number).

```
 IDEG-----> major command for MDEX and PDEX
aMDEXnn GRP=ccccccccc
-or-
aPDEXnn GRP=ccccccccc
```

  Figure 22. MDEX/PDEX commands format

- Use the command argument field or the GRP= keyword to limit the scope of the Bottleneck Analysis display to specific execution state groups. To see only certain transactions, enter a Bottleneck Analysis group number from 1 to 30 (or the current **MAXG** value) with the **MDEX** or **PDEX** command. For more information about transaction groups or the **MAXG** command, see the *IBM OMEGAMON for IMS Realtime Commands Reference*.

  If you leave both the command argument field and the GRP= keyword blank, Bottleneck Analysis displays the execution states of all IMS transactions that are not excluded by the **BMPX** or **DOPT** minor commands of **IDEG**.

- Use the command label field to restrict the display to specific execution states. Valid command label specifications are as follows:

blank
Displays all execution state analyses.

D
Displays database I/O waits only.

I
Displays IMS internal waits only.

M
Displays z/OS waits only.

S
Displays scheduling waits only.

O
Displays output waits.

E
Displays external subsystem waits.

Examples of MDEX and PDEX output in Bottleneck Analysis displays

At first glance, the displays that **MDEX** and **PDEX** produce look very much alike. Both displays show activity for the execution states over both a short-term interval and a long-term interval. The short-term interval is the most recent portion of the long-term interval.

The first line of the display is a subtotal that covers the entire group. The individual execution states that comprise the subtotal follow, subject to the threshold specified by using the **MTHR** minor command for **MDEX**, or the threshold percentage by using the **THRS** minor command for **PDEX**.

The subtotals by group are not enclosed in parentheses, while the numbers for the individual execution states are, to visually distinguish them.

Adjacent to each execution state name, a notation in parentheses indicates whether this state is a running, a competing, or a non-competing execution state. Competing and non-competing states are mutually exclusive, while running states are a correct subset of competing ones.

Example of MDEX output

Figure 23 on page 43 shows a typical **MDEX** display, where:

- The command specified is **MDEX**, which displays the average count of transactions in each execution state.
- The label field does not contain a selection character, so all types of execution states are eligible for display.
- The argument field does not contain a transaction group number and the GRP= keyword is not specified, so **MDEX** displays data about all transactions.
- The Bottleneck Analysis component last cleared the long-term counters 1 minute and 41 seconds before **MDEX** generated this display (**Elapsed time= 1:41 MN**).
- The **MDEX** display includes information about *all* transactions. The last line of the display indicates **Avg. Total Trans:**, rather than just **Avg. Trans Executing:** or **Avg. Trans Competing**:.

  Whether the **DOPT** option is COMP or EXEC, it is important to look at the **Avg. Total Trans:** line to get an indication of the load on the system.
- The display threshold that is set by **MTHR** is 0.0, so all nonzero execution states display.
- The scaling factor that is set by **SCAL** is 1, so **MDEX** multiples all values that appear by 1, and they are thus true values.

```
 MDEX                      ------Short Term -----       ------Long Term ------
+ (Elapsed time= 1:41 MN)   0_______ 10_______ 20        0_______ 10_______ 20
+ Using CPU:            3.0|--> .    .    .    .|    1.7|>   .    .    .    .|
+  Using CPU In APPL   (1.1)|>   .    .    .    .|   (.4)|>   .    .    .    .|
+  Using CPU In IMS    (1.9)|>   .    .    .    .|  (1.3)|>   .    .    .    .|
+ Scheduling Waits:        0|    .    .    .    .|     .3|>   .    .    .    .|
+  Wait For GU           (0)|>   .    .    .    .|   (.1)|>   .    .    .    .|
+  Wait For MPP          (0)|>   .    .    .    .|   (.1)|>   .    .    .    .|
+  Wait For Resched      (0)|>   .    .    .    .|   (.1)|>   .    .    .    .|
+ Database I/O Waits     1.9|>   .    .    .    .|    1.2|>   .    .    .    .|
+  DI21PART            (1.1)|>   .    .    .    .|   (.7)|>   .    .    .    .|
+  BE3PART              (.8)|>   .    .    .    .|   (.5)|>   .    .    .    .|
+ z/OS Waits:             .2|>   .    .    .    .|     .1|>   .    .    .    .|
+  CPU Wait (MPP/BMP)   (.2)|>   .    .    .    .|   (.1)|>   .    .    .    .|
+ IMS Activity:           .2|>   .    .    .    .|     .0|>   .    .    .    .|
+  ISWITCHED to CTL     (.2)|>   .    .    .    .|   (.0)|>   .    .    .    .|
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ Avg Trans Executing  (5.4)|---->    .    .    .|  (3.2)|--> .    .    .    .|
+ Avg Trans Competing  (5.4)|---->    .    .    .|  (3.2)|--> .    .    .    .|
+ Avg Trans Not Comp     (0)|    .    .    .    .|    (0)|    .    .    .    .|
+ Avg Total Trans:     (5.4)|---->    .    .    .|  (3.2)|--> .    .    .    .|
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ Display threshold (MTHR) is .0 / Scaling factor (SCAL) is 1
```

Figure 23. MDEX command output

Example of PDEX output

Figure 24 on page 43 shows a typical **PDEX** display, where:

- The command that is specified is **PDEX**, which displays the percentage of time an average transaction was in each execution state.
- The label field does not contain a selection character, so all types of execution states are eligible for display.
- The argument field does not contain a transaction group number and the GRP= keyword is not specified, so data about all transactions displays.
- The Bottleneck Analysis component last cleared the long counters 1 minute and 41 seconds before **PDEX** generated this display (**Elapsed time= 1:41 MN**).
- The **PDEX** display is limited to executing transactions, indicating that the **DOPT** option was set to EXEC. (“Performing Bottleneck Analysis data sampling by transaction type” on page 20 describes the **DOPT** command.) The last line of the display indicates **Avg. Trans Executing:**, rather than **Avg. Trans Competing:** or **Avg. Total Trans:**.

  Whether the **DOPT** option is COMP or EXEC, it is important to look at the last line to get an indication of the load on the system.

```
 PDEX                      ------Short Term %----      ------ Long Term %-----
+ (Elapsed time= 1:41 MN) % 0_______ 50_______100     % 0________ 50_______100
+ Using CPU:              46.0|-----====>    .    .|  44.0|------====>    .    .|
+  Using CPU In APPL    (20.0)|--->.    .    .    .|(12.0)|->  .     .    .    .|
+  Using CPU In IMS     (26.0)|------>  .    .    .|(32.0)|------=>  .    .    .|
+ Scheduling Waits:        3.0|>   .    .    .    .|   3.0|>   .     .    .    .|
+  Wait for GU           (1.0)|>   .    .    .    .| (1.0)|>   .     .    .    .|
+  Wait for MPP          (1.0)|>   .    .    .    .| (1.0)|>   .     .    .    .|
+  Wait for Resched      (1.0)|>   .    .    .    .| (1.0)|>   .     .    .    .|
+ Database I/O Wait       36.0|------>  .    .    .|  38.0|------>    .    .    .|
+  DI21PART             (20.0)|--->.    .    .    .|(24.0)|---->     .    .    .|
+  BE3PART              (16.0)|--> .    .    .    .|(14.0)|--> .     .    .    .|
+ IMS Activity            15.0|->  .    .    .    .|   8.0|>   .     .    .    .|
+  ISWITCHED to CTL      (7.0)|>   .    .    .    .| (2.0)|>   .     .    .    .|
+  IRLM Conflict Wait    (8.0)|>   .    .    .    .| (6.0)|>   .     .    .    .|
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ Avg. Trans Executing:                 5.4                          3.2
```

Figure 24. PDEX command output

Setting the display threshold for the MDEX command

The Bottleneck Analysis component recognizes more IMS transaction execution states than the **MDEX** command can display on any model 3270 screen. To reduce the size of the display, use the **MTHR** command to specify a threshold value.

About this task

The **MDEX** command does not display any state whose short- and long-term average counts are both less than the value specified by using the **MTHR** command.

Procedure

- Enter the **MTHR** minor command in command mode as follows:

```
MTHR nnnn
```

Where *nnnn* is 0 - 9999 (9999 represents a threshold value of 999.9 transactions.) The default for **MDEX** is 0. All nonzero transaction states are displayed. If you omit *nnnn*, **MTHR** displays the current threshold value.

```
IDEG   >> Elapsed time= 3:15 MN, #samples(short)=352, #samples(long)=352 <<
mthr  3   >> .3 is MDEX display threshold <<
```

Figure 25. MTHR command output

What to do next

The **MDEX** command cannot be forced to display all possible transaction states. You can change the default permanently on the MINIMUM_COUNT_THRESHOLD keyword in the KIPGLB member that is used by the OMEGAMON Classic address space. Refer to the *IBM OMEGAMON for IMS on z/OS: Planning and Configuration Guide* for details.

Setting the scaling factor for the MDEX command

The **MDEX** command calculates average transaction counts for each of the various execution states to only one decimal place, that is, 0.1 transactions so that low-volume systems can experience many counts that approximate to zero. Use the **SCAL** command to specify a scaling factor multiplier, which is multiplied against the value before the transaction average is computed. You can see in more detail where transactions are spending time.

Procedure

- Enter the **SCAL** minor command in command mode as follows:

```
SCAL nnn
```

Where *nnn* is 1 - 100. If you do not provide a value, the Bottleneck Analysis component uses the setting for SCALING_FACTOR in the KIPGLB member that is used by the OMEGAMON Classic address space. The default is 1.

The following figure shows the **SCAL** command in combination with the **MTHR** command.

```
IDEG   >> Elapsed time= 2:49 MN, #samples(short)=372, #samples(long)=372 <<
MTHR   >> .0 is MDEX display threshold <<
SCAL   >> 10 is MDEX display scale value <<
```

Figure 26. MTHR and SCAL commands output

Example

If you specify a scaling factor of 10, an average count of 0.03 displays as 0.3 instead of being rounded down to 0.0.

Setting the display threshold for the PDEX command

The Bottleneck Analysis component can detect and display many execution states, so the **PDEX** display can become large. Use the **THRS** minor command to set a threshold that suppresses insignificant states on the **PDEX** display.

About this task

If the percentage of total transaction time of a particular state in both the short- and long-term interval is less than the **THRS** threshold, **PDEX** does not include it in the display. In this case, the percentages that do display might total less than 100%.

Procedure

- Issue the **THRS** minor command as follows:

  ```
  THRSnn
  ```

  The **THRS** command accepts values 0 - 99. If you do not specify a value, **THRS** displays the current setting. If you specify 0 (the default), **PDEX** does not display transaction states that were never seen, because specifying 0 generates a large, useless display. Yet, some statistics might still display as zero because they were rounded down to zero. Only true zero statistics are displayed by the **PDEX** command if they are for the short interval and the transaction state in the long interval is nonzero.

```
IDEG   >> Elapsed time= 4:19 MN, #samples(short)=458, #samples(long)=458 <<
thrs   >> 0 is PDEX wait percent threshold <<
```

Figure 27. THRS command output

What to do next

You can change the **THRS** default permanently on the MINIMUM_PERCENT_THRESHOLD keyword in the KIPGLB member that is used by the OMEGAMON Classic address space. Refer to the *IBM OMEGAMON for IMS on z/OS: Planning and Configuration Guide* for details.

Viewing the transaction time spent in execution states

You can view a graph that represents the percentage of time an average transaction that is spent in each of the execution states that the Bottleneck Analysis collector monitors.

Procedure

From the Bottleneck Analysis menu, select option **A**, **EXECUTING** and press Enter.

A screen similar to the one shown in Figure 28 on page 46 displays.

```
 _________________ KDIEXEC  VTM     OI-II    V530./C I9IC 04/30/15 13:41:16    B
> Help PF1               Back PF3               Up PF7                Down PF8
================================================================================
>                   Factors Affecting Executing Transactions

> To display information about a specific group, enter the group number
> directly after PDEX below.

> Enter D, I, M, or S directly to the left of PDEX to display database I/O,
> IMS internal, MVS, or scheduling waits only.

 IDEG   >> Elapsed time=19:13 MN, #samples(short)=489, #samples(long)=2214 <<
>dopt  EXEC  >> Only Executing Transactions Will Be Analyzed <<

 pdex                        ------Short Term %----      ------ Long Term %-----
+ (Elapsed time= 1:41 MN) % 0_______ 50_______100     % 0________ 50_______100
+ Using CPU:              56.0|-----====>     .    .|  54.0|------====>    .    .|
+   Using CPU In APPL   (20.0)|--->.     .     .    .|(12.0)|->  .     .    .    .|
+   Using CPU In IMS    (36.0)|------>  .     .    .|(42.0)|------=>  .    .    .|
+ Database I/O Wait       36.0|------>  .     .    .|  38.0|------>   .    .    .|
+   DI21PART            (20.0)|--->.     .     .    .|(24.0)|---->     .    .    .|
+   BE3PART             (16.0)|--> .     .     .    .|(14.0)|--> .     .    .    .|
+ IMS Activity             8.0|->  .     .     .    .|   8.0|->  .     .    .    .|
+   ISWITCHED to CTL     (4.0)|>   .     .     .    .| (2.0)|>   .     .    .    .|
+   IRLM Conflict Wait   (4.0)|>   .     .     .    .| (6.0)|>   .     .    .    .|
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+ Avg. Trans Executing:                 5.4                        3.2
================================================================================
```

Figure 28. Executing Transactions screen

Note: If you are using the Bottleneck Analysis component in a Database Control (DBCTL) environment, your screen might look different from the one in Figure 28 on page 46. Because transactions do not exist in a DBCTL environment, fields that relate to transactions are not displayed. See "DBCTL environment" on page 51 for a list of commands that are not applicable in a DBCTL environment.

See "Setting Bottleneck Analysis display commands (MDEX/PDEX)" on page 41 for more information about Bottleneck Analysis displays.

Bottleneck Analysis collector abend display

The **ABCD** command displays important Bottleneck Analysis information that you can provide to IBM Software Support. If the Bottleneck Analysis collector abends, for example, the Bottleneck Analysis component saves PSW and register information.

The following figure shows typical ABCD command output.

```
 IDEG   >> Elapsed time= 4:19 MN, #samples(short)=458, #samples(long)=458 <<
 ABCD   >> OI206: Collector has not ABENDed <<
```

Figure 29. ABCD command output

A warning message is displayed if the collector abends when you issue the **IDEG** major command. The collector typically recovers from errors that the Bottleneck Analysis component encounters, for example, an 0C4 abend, or loops due to a chain moved. Report any abends.

Average transaction counts statistics in Bottleneck Analysis displays

In Bottleneck Analysis displays, you can view the average number of either total, executing, or competing transactions. The display depends on the setting of the **DOPT** command for both the short-term and the long-term interval.

At the bottom of the **MDEX** display, Bottleneck Analysis displays the average number of total, executing, competing, and non-competing transactions for both the short-term and the long-term intervals. While they are not statistics about execution states, these numbers can be valuable.

The **PDEX** profile of an IMS system can remain the same or even improve over normal operation. DEXAN does not see all stages in the life of an IMS transaction. Specifically, it does not see transit time in the TP network.

If network delays become a problem, for example, if a link fails and network traffic reroutes over a much slower backup link, the load on the central IMS system decreases. When the load decreases, the **PDEX** profile improves, that is, shifts towards using the processor because of the decreased contention for system resources. However, the average transaction counts change. The total transaction count might not change very much because of the large number of transactions that might be in the queue for BMPs or MPPs that do not reach their threshold, but the average competing transaction count might plummet, reflecting the reduced workload that reaches the system. When the load on the central IMS system decreases, suspect network problems.

Chapter 6. Reference: Bottleneck Analysis command summary

The following information provides you with a quick reference summary of the Bottleneck Analysis and transaction group commands. It also identifies restrictions that apply to a Database Control (DBCTL) environment.

Bottleneck Analysis commands

The following sections list each Bottleneck Analysis command.

Major command

IDEG
Controls execution of Bottleneck Analysis. A hyphen (**-**) in column 1 attaches the Bottleneck Analysis collector.

Minor commands of IDEG

ABCD
Data collector ABEND completion code.

BEGN
Begins Bottleneck Analysis data collector.

BMPX
Displays the current setting of BMP switch.

BMPXON
Data collector ignores BMP activity.

BMPXOFF
Data collector does not ignore BMP activity.

CLRL
Specifies the long-term clearing interval in minutes.

CLRS
Specifies the short-term clearing interval in minutes.

DBSW
Displays current setting of database I/O switch.

DBSWON
Collects and displays database I/O values by individual database name.

DBSWOFF
Collects and displays total database I/O values only.

DOPT
Displays information about competing, running, or all transactions.

DTCH
Detaches collector subtask.

END
Stops data collector.

MDEX
Displays Bottleneck Analysis results by count (group *nn*).

MTHR
Specifies display threshold for MDEX.

PDEX
Displays Bottleneck Analysis results by percentage (group *nn*).

RESM
Resumes collector.

SCAL
Specifies scaling factor for MDEX.

STIM
Sets the collection interval in tenths of a second.

SUSP
Suspends collector.

THRS
Suppresses execution states that occur less than *nn* percent of the time on PDEX display.

To limit the **MDEX** and **PDEX** displays to a specific execution state group, enter one of the following arguments in the label field (column 1):

blank
All execution state analyses

D
DB I/O waits

I
IMS waits

M
z/OS waits

S
Scheduler waits

O
Output waits

E
External subsystem waits

Transaction group commands

The following section lists each transaction group command.

GLBL*cc*
Displays or changes current suffix of KIPGLBcc member; entering *cc* causes a new globals member to be loaded.

MAXG*nn*
Dynamically controls the number of transaction groups for IMS, or PSB groups for DBCTL, that OMEGAMON Classic supports.

a*SETG*nn ccc
Displays or changes contents of a transaction, logical terminal, or node group. In this command, *nn* is the group number (enter 99 for all groups); *ccc* is the entry specification (PSB=*xxx*, TRAN=*xxx*, TERM=*nnn*, NODE=*xxx*, or CLASS=*nnn*); *a* is the function that is invoked. The functions are:

blank
Lists the contents of group.

A
Adds an entry to group definition.

C
Creates a group.

D
Deletes an entry from group.

L
Lists the contents of group (default).

X
Deletes all group contents.

DBCTL environment

You cannot use some commands in a DBCTL environment.

The following commands are not applicable in a DBCTL environment:

- DOPT ccccc
- AUTO
- TRAN=, TERM=, CLASS= entry specifications of SETG

The following execution state groups are not applicable in a DBCTL environment:

- IMS scheduling waits
- Output waits
- External subsystem waits

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully.OMEGAMON monitoring products support several user interfaces. Product functionality and accessibility features vary according to the interface.

The major accessibility features in this product enable users in the following ways:

- Use assistive technologies, such as screen-reader software and digital speech synthesizer, to hear what is displayed on the screen. Consult the product documentation of the assistive technology for details on using those technologies with this product.
- Operate specific or equivalent features using only the keyboard.
- Magnify what is displayed on the screen.

In addition, the product documentation was modified to include the following features to aid accessibility:

- All documentation is available in both HTML and convertible PDF formats to give the maximum opportunity for users to apply screen-reader software.
- All images in the documentation are provided with alternative text so that users with vision impairments can understand the contents of the images.

Some content presented in IBM Documentation might not yet be in a format that a screen reader can process. If you need help, contact ibmkc@us.ibm.com.

Interface information

The Tivoli Enterprise Portal interface offers the greatest range of functionality, but is not entirely accessible. The OMEGAMON enhanced 3270 user interface offers more limited functionality, but is entirely accessible. (The enhanced 3270 user interface supports all the accessibility features supported by your emulator. If you are using IBM Personal Communications, you can find information about its accessibility features in the Using Emulator Sessions topic. If you are using a third-party emulator, see the documentation for that product for accessibility information.)

The OMEGAMON ("classic") interface uses an ISPF style interface. Standard and custom PF Key settings, menu options, and command-line interface options allow for short cuts to commonly viewed screens. While basic customization options allow for highlights and other eye-catcher techniques to be added to the interface, the customization options are limited.

Related accessibility information

Some content presented in IBM Documentation might not yet be in a format that a screen reader can process. If you need help, contact ibmkc@us.ibm.com.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online
Go to the IBM Software Support site at http://www.ibm.com/software/support/probsub.html and follow the instructions.

Troubleshooting Guide
For more information about resolving problems, see the product's Troubleshooting Guide.

Product legal 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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan, Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

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 might 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation 2Z4A/101 11400 Burnet Road Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be

the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

If you are viewing this information in softcopy form, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions:

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of

these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user’s user name for purposes of session management, authentication, and single sign-on configuration. These cookies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Index

Special Characters

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W

X

Z