Rational Developer for System z Version 7.5

Developer for System z and WLM

Tips on how to configure Workload Manager for use with Developer for System z

SC27-3605-00

Onno Van den Troost

Note: Before using this document, read the general information under *Documentation notices for IBM™ Rational™ Developer for System z.*

May 2010

Order publications by phone or fax. IBM™ Software Manufacturing Solutions takes publication orders between 8:30 a.m. and 7:00 p.m. eastern standard time (EST). The phone number is (800) 879-2755. The fax number is (800) 445-9269. Faxes should be sent Attn: Publications, 3rd floor.

You can also order publications through your IBM™ representative or the IBM™ branch office serving your locality. Publications are not stocked at the address below.

IBM™ welcomes your comments. You can send your comments by mail to the following address:

IBM Corporation
Attn: Information Development Department 53NA
Building 501 P.O. Box 12195
Research Triangle Park NC 27709-2195
USA

You can fax your comments to: 1-800-227-5088 (US and Canada)

Contents

Figures

Tables

Preface

About this document

This document discusses z/OS Workload Manager (WLM), and how IBM Rational Developer for System z Version 7.5 can be managed by WLM.

Who should read this document

This document is intended for system programmers who have been tasked to define workload classification for Rational Developer for System z Version 7.5. To use this guide, you need to be familiar with z/OS Workload Manager (WLM).

Summary

One of the strengths of the System z platform and the z/OS operating system is the ability to run different types of workload at the same time within one z/OS image or across multiple images. The function that makes this possible is dynamic workload management, which is implemented in the Workload Manager (WLM) component of the z/OS operating system.

Unlike traditional z/OS applications, Developer for System z is not a monolithic application that can be identified easily to WLM. Developer for System z consists of several components that interact to give the client access to the host services and data. These different tasks communicate with each other, which implies that the actual elapsed time becomes important to avoid time-out issues for the connections between the tasks.

As a result, Developer for System z tasks should be placed in high-performance service classes, or in moderate-performance service classes with a high priority. Velocity goals are advised for all critical tasks.

You should revise, and possibly update, your current WLM goals. This is especially true for traditional MVS shops new to time-critical OMVS workloads.

Table 1 lists the address spaces that are used by Developer for System z. z/OS UNIX will substitute "x" in the "Task Name" column by a random 1-digit number.

Description	**Task name**	**Workload**
JES Job Monitor	JMON	STC
Lock daemon	LOCKD	STC
RSE daemon	RSED	STC
RSE thread pool	RSEDx	OMVS
ISPF Client Gateway (TSO Commands service and SCLMDT)	<userid>x	OMVS
TSO Commands service (APPC)	FEKFRSRV	ASCH
CARMA (batch)	CRA<port>	JES
CARMA (crastart)	<userid>x	OMVS
CARMA (ISPF Client Gateway)	<userid> and <userid>x	OMVS
MVS build (batch job)	*	JES
z/OS UNIX build (shell commands)	<userid>x	OMVS
z/OS UNIX shell	<userid>	OMVS
File Manager task	<userid>x	OMVS
Application Deployment Manager	CICSTS	CICS

Table 1. WLM workloads

Understanding Workload Manager

One of the strengths of the System z platform and the z/OS operating system is the ability to run different types of workload at the same time within one z/OS image or across multiple images. The function that makes this possible is dynamic workload management, which is implemented in the Workload Manager (WLM) component of the z/OS operating system.

The installation (user) classifies the work running on the z/OS operating system in distinct service classes and defines goals for them that express the expectation of how the work should perform. WLM uses these goal definitions to manage the work across all systems of a sysplex environment.

- The identification of work requests is supported by the middleware and the operating system. Work requests tell WLM when a new unit of work enters the system and when it leaves the system. WLM provides constructs to separate the work into distinct classes so it can deal with different types of work running on the system.
- Contention can occur when multiple units of work want to use the system resources. These are the CPUs, the I/O devices, and storage, but also software constructs, such as processes or address spaces, which provide the capability to execute programs and serialization points that allow the programs to access resources or serialized control areas. WLM monitors these resources to be able to understand how many resources the work requires or will wait for.
- The work classification and the WLM observation of how the work uses the resources provide the base to manage the system. This management is done based on the goals that the installation defines for the work. After classifying the work into distinct classes, the installation associates a goal and business priority with each class, which determines how much service the work in the class is able to receive.

WLM components

All the business performance requirements of an installation are stored in a service policy. There is one active service policy for the entire sysplex, but WLM can dynamically switch between service policies.

A service policy includes:

- Workloads: Arbitrary names used to group various service classes together for reporting and accounting purposes.
- Service classes: Where you define one or more periods which specify the performance goal, business importance and period duration for a specific type of work.
- Report classes: Aggregate set of work for reporting purpose.

- Performance goals: The desired level of service that WLM uses to determine the amount of resource to give to a unit of work.
- Classification rules and classification groups: Used to assign the incoming work to a service class and, if needed, to a report class.

There are more WLM components and service policy elements than listed here, but these are of no importance in the following discussion.

A WLM ISPF application, `SYS1.SBLSCLI0(IWMARIN0)`, can be used to manage service policies.

The Resource Measurement Facility (RMF™) Workload Activity Report groups performance data by workload and also by service class periods within workloads, giving you data to base your WLM service class definitions upon.

Sampling dispatchable unit states

In order to enforce goals and to track the performance of a sysplex, WLM samples the states of dispatchable units of work every 250 milliseconds.

Within each sample, a unit of work can be in one of the following states:

- *Using*, when using CPU or DASD I/O
- *Delayed*, when delayed by CPU, I/O, storage, or resource usage queues
- *Idle*, when without work (such as TSO or OMVS without transactions, an initiator without a job, APPC wait, or a server in STIMER wait)
- *Unknown*, when delayed by a non-tracked WLM resource (such as ENQ or operator) or idle for a reason other than those listed under the idle state above
- *Quiesced*, when the unit of work is quiesced by the RESET operator command

Service classes

A service class describes a group of work within a workload with similar performance goals, resource requirements, or business importance. A service class consists of one or more periods. A period specifies a goal, an importance, and a duration.

Period duration

Duration is the amount of service units that a period can consume before going on to the next period (which has a new goal). A service unit is a normalized metric used by z/OS to measure the CPU consumption by dispatchable units of work.

Business importance

The importance for a service class defines how important it is to achieve the performance goal for that service class. At runtime, the workload management component manages

workload distribution and allocation of resources to competing workloads. High priority workloads get guaranteed consistent results, for example, response time, throughput, and so forth.

Performance goal

A goal expresses the expectation of how the work in the service class period should perform. There are different goal types:

- *Average response time*
  Average response time is the expected amount of time required to complete the work in this period. This only includes the time spent on z/OS. For example, it will not include network time.
- *Percentile response time*
  Percentile response time is the percentage of work in this period that should complete within the defined response time. For example, 80% of the transactions should end in 0.5 seconds.
- *Velocity*
  Velocity goals can be used where it is not possible to use a response time goal. By monitoring how often individual work and the service class were delayed (for WLM-managed resources), WLM is able to express their speed in the system, dubbed *execution velocity*, as the amount of delay for work that is ready to run. The result is a number between 0 and 100, where 100 means that the work had no delay at all. A velocity goal defines the acceptable amount of delay for the work.
- *Discretionary*
  Work for which no concrete goal must be achieved is assigned to service classes with a discretionary goal. This tells WLM to just do the best it can, and that the work should only run when service classes with higher importance do not need the resources to achieve their goals.

Understanding Developer for System z

The Developer for System z host consists of several components that interact to give the client access to the host services and data. Understanding the design of these components can help you make the correct configuration decisions.

Component overview

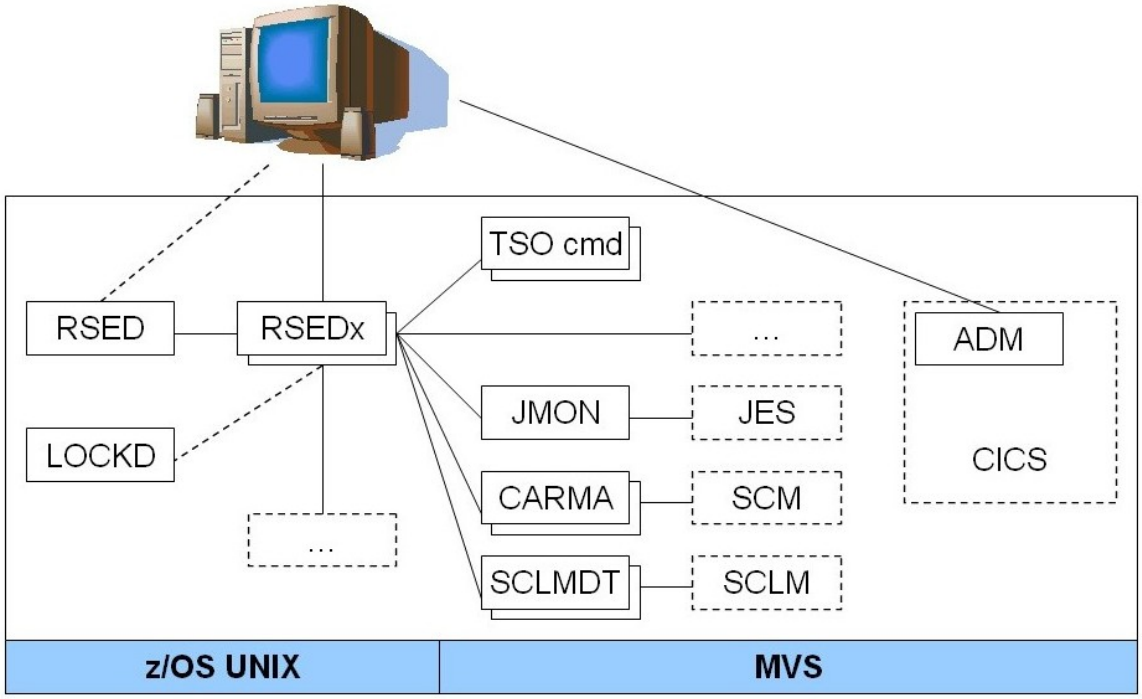


Figure 1. Component overview

Figure 1 shows a generalized overview of the Developer for System z layout on your host system.

- Remote Systems Explorer (RSE) provides core services, such as connecting the client to the host and starting other servers for specific services. RSE consists of two logical entities:
  - RSE daemon (RSED), which manages connection setup. RSE daemon is also responsible for running in single server mode. To do so, RSE daemon creates one or more child processes known as RSE thread pools (RSEDx).
  - RSE server, which handles individual client request. An RSE server is active as a thread inside a RSE thread pool.
- Lock Daemon (LOCKD) provides tracking services for data set locks.
- TSO Command service (TSO cmd) provides a batch-like interface for TSO and ISPF commands.
- JES Job Monitor (JMON) provides all JES related services.
- Common Access Repository Manager (CARMA) provides an interface to interact with Software Configuration Managers (SCMs), such as CA Endevor.

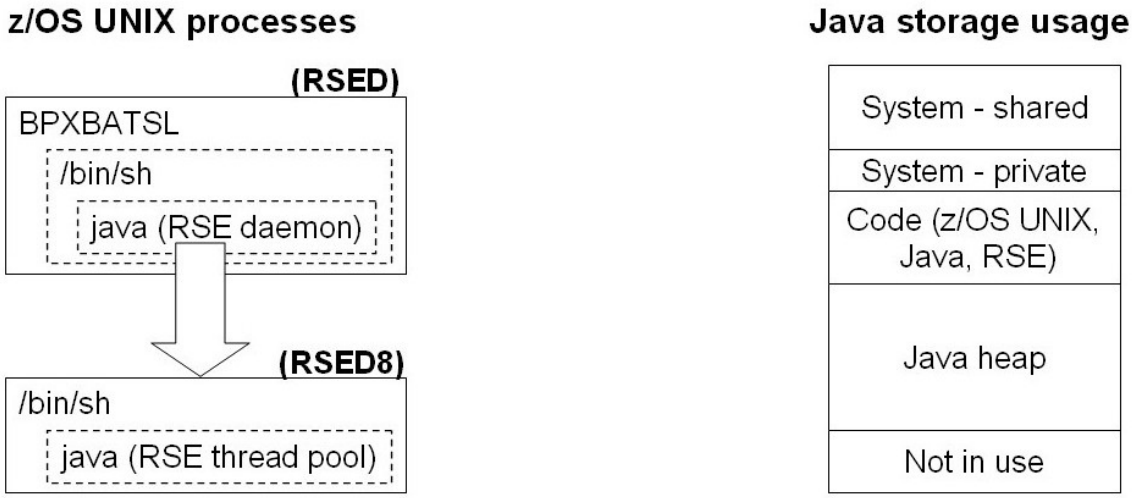
- SCLM Developer Toolkit (SCLMDT) provides an interface to enhance and interact with SCLM.
- Application Deployment Manager (ADM) provides various CICS related services.

More services are available, which can be provided by Developer for System z itself or corequisite software.

The description in the previous paragraph and list shows the central role assigned to RSE. With few exceptions, all client communication goes through RSE. This allows for easy security related network setup, as only a limited set of ports are used for client-host communication.

To manage the connections and workloads from the clients, RSE is composed of a daemon address space, which controls thread pooling address spaces. The daemon acts as a focal point for connection and management purposes, while the thread pools process the client workloads. Based upon the values defined in the `rsed.envvars` configuration file, and the amount of actual client connections, multiple thread pool address spaces can be started by the daemon.

RSE as a Java application




```
JOBNAME  Status                    PID      PPID  Command
RSED     FILE SYS KERNEL WAIT  50331904         1  BPXBATSL
RSED     WAITING FOR CHILD     67109114  50331904  /bin/sh ...
RSED     FILE SYS KERNEL WAIT  50331949  67109114  java ...

RSED8    WAITING FOR CHILD          307  50331949  /bin/sh ...
RSED8    FILE SYS KERNEL WAIT       308       307  java ...
```

Figure 2. RSE as a Java application

Figure 2 shows a basic view of resource usage (processes and storage) by RSE.

RSE is a Java application, which means that it is active in the z/OS UNIX environment. This allows for easy porting to different host platforms (like Linux on System z) and

straightforward communication with the Developer for System z client, which is also a Java application (based on the Eclipse framework). Therefore, basic knowledge of how z/OS UNIX and Java work is very helpful when you try to understand Developer for System z.

In z/OS UNIX, a program runs in a process, which is identified by a PID (Process ID). Each program is active in its own process, so invoking another program creates a new process. The process that started a process is referenced with a PPID (Parent PID). The new process is called a child process. The child process can run in the same address space or it can be spawned (created) in a new address space. A new process that runs in the same address space can be compared to executing a command in TSO, while the spawning one in a new address space is similar to submitting a batch job.

Note that a process can be single- or multi-threaded. In a multi-threaded application (such as RSE), the different threads compete for system resources as if they were separate address spaces (with less overhead).

Mapping this process information to the RSE sample in Figure 2, we get the following flow:

1. When the RSED task is started, it executes `BPXBATSL`, which invokes z/OS UNIX and creates a shell environment - PID 50331904.
2. In this process, the `rsed.sh` shell script is executed, which runs in a separate process (/bin/sh) - PID 67109114.
3. The shell script sets the environment variables defined in `rsed.envvars` and executes Java with the required parameters to start the RSE daemon - PID 50331949.
4. RSE daemon will spawn off a new shell in a child process (RSED8) - PID 307.
5. In this shell, the environment variables defined in `rsed.envvars` are set and Java is executed with the required parameters to start the RSE thread pool - PID 308.

Java applications, such as RSE, do not allocate storage directly, but use Java memory management services. These services, like allocating storage, freeing storage, and garbage collection, work within the limits of the Java heap. The minimum and maximum size of the heap is defined (implicitly or explicitly) during Java startup.

This implies that getting the most out of the available address space size is a balancing act of defining a large heap size while leaving enough room for z/OS to store a variable amount of system control blocks (dependant on the number of active threads).

Task owners

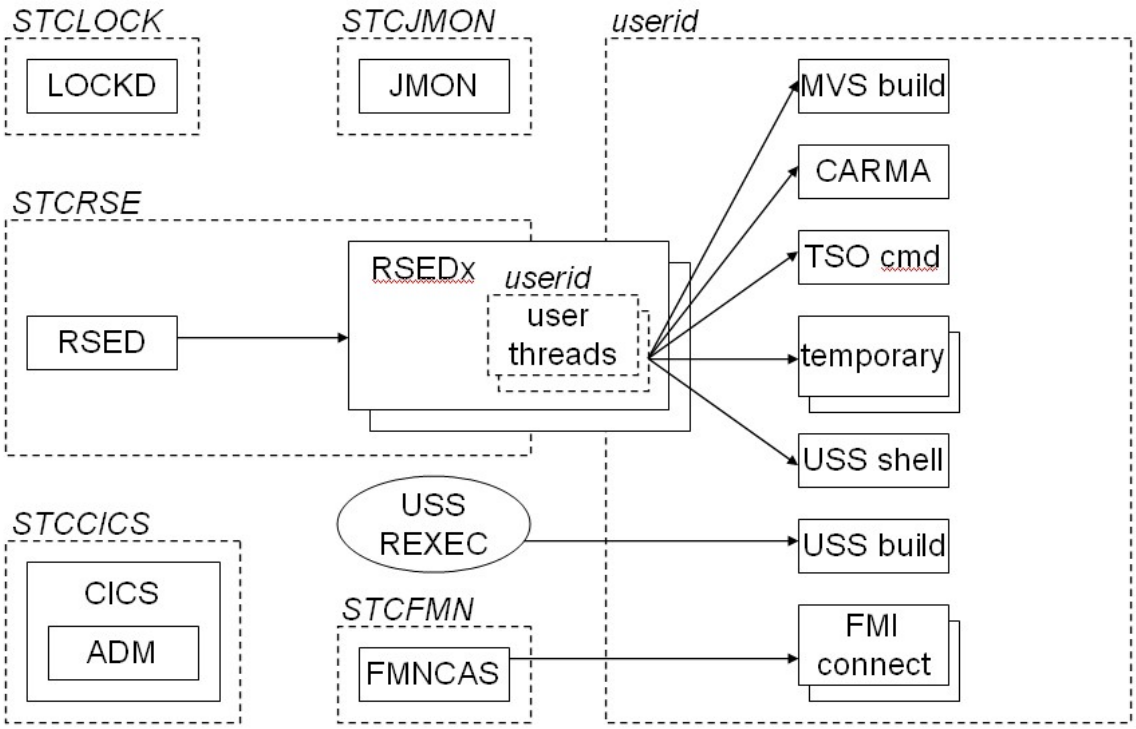


Figure 3. Task owners

Figure 3 shows a basic overview of the owner of the security credentials used for various Developer for System z tasks.

The owner of a task can be roughly divided into 2 sections. Started tasks are owned by the user ID that is assigned to the started task in your security software. All other tasks, with the RSE thread pools (RSEDx) as exception, are owned by the client user ID.

Figure 3 shows both the Developer for system z started tasks (LOCKD, JMON and RSED), and sample started tasks and system services that Developer for System z communicates with. Application Deployment Manager (ADM) is active inside a CICS region. FMNCAS is the File Manager started task. The USS REXEC tag represents the z/OS UNIX REXEC (or SSH) service.

RSE daemon (RSED) creates one or more RSE thread pool address spaces (RSEDx) to process client requests. Each RSE thread pool supports multiple clients and is owned by the same user as the RSE daemon. Each client has his own threads inside a thread pool, and these threads are owned by the client user ID.

Depending on actions done by the client, one or more additional address spaces can be started, all owned by the client user ID, to perform the requested action. These address spaces can be an MVS batch job, an APPC transaction or a z/OS UNIX child process.

Note that a z/OS UNIX child process is active in a z/OS UNIX initiator (BPXAS) and it shows up as a started task in JES.

The creation of these address spaces is most often triggered by a user thread in a thread pool, either directly or by using system services like ISPF. But the address space could also be created by a third party. For example, File Manager will start a new address space for each data set (or member) it has to process on behalf of Developer for System z. z/OS UNIX REXEC (or SSH) are involved when starting builds in z/OS UNIX.

The user-specific address spaces end at task completion or when an inactivity timer expires. The started tasks remain active. The address spaces listed in Figure 3 remain in the system long enough to be visible, but you should be aware that due to the way z/OS UNIX is designed, there are also several short-lived temporary address spaces.

Connection flow

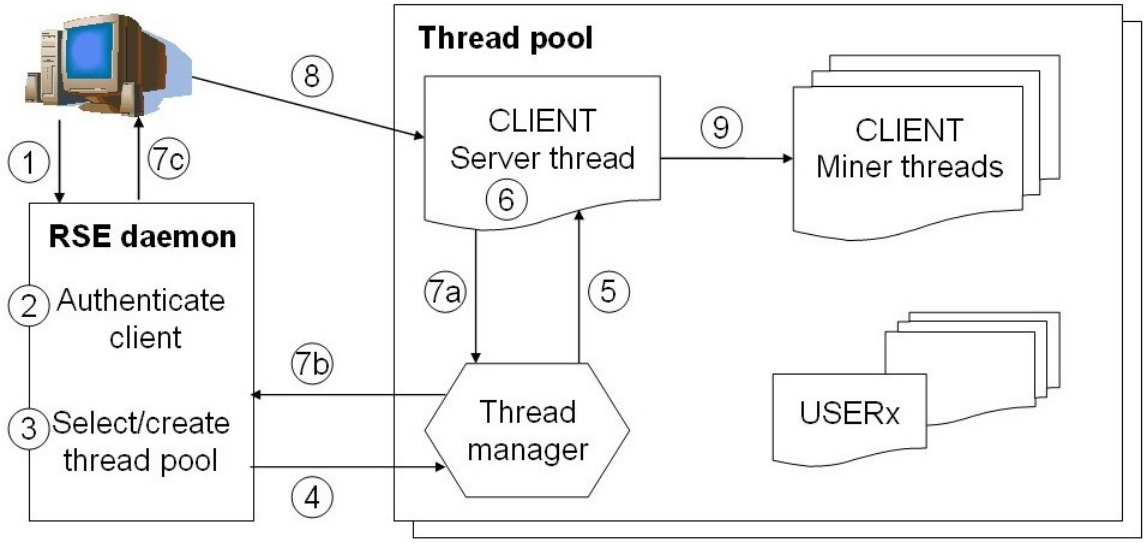


Figure 4. Connection flow

Figure 4 shows a schematic overview of how a client connects to the host using Developer for System z.

1. The client logs on to the daemon (port 4035).
2. RSE daemon authenticates the client, using the credentials presented by the client.
3. RSE daemon selects an existing thread pool or starts a new one if all are full.
4. RSE daemon passes the client user ID on to the thread pool.
5. The thread pool creates a client-specific RSE server thread.
6. The client server thread binds to a port for future client communication.
7. The client server thread returns the port number for the client to connect to.
8. The client disconnects from RSE daemon and connects to the provided port number.

9. The client server thread starts other user-specific threads (miners). These threads provide the user-specific services requested by the client.

The previous description shows the thread-oriented design of RSE. Instead of starting an address space per user, multiple users are serviced by a single thread pool address space. Within the thread pool, each miner (a user-specific service) is active in its own thread with the user's security context assigned to it, ensuring a secure setup. This design accommodates large number of users with limited resource usage, but does imply that each client will use multiple threads (16 or more, depending on the performed tasks).

WLM and Developer for System z

Unlike traditional z/OS applications, Developer for System z is not a monolithic application that can be identified easily to WLM. Developer for System z consists of several components that interact to give the client access to the host services and data. As described in *Understanding Developer for System z*, some of these services are active in different address spaces, resulting in different WLM classifications.

Workload classification

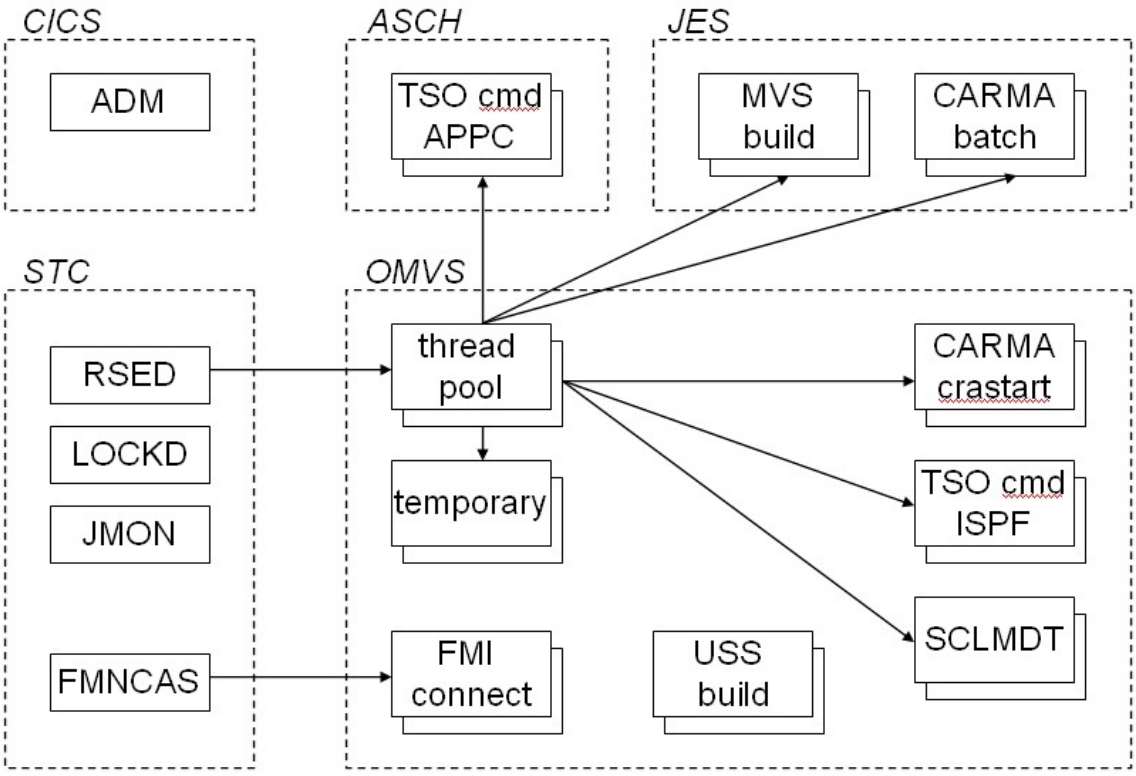


Figure 5. WLM classification

Figure 5 shows a basic overview of the subsystems via which Developer for System z workloads are presented to WLM.

Application Deployment Manager (ADM) is active within a CICS region, and will therefore follow the CICS classification rules in WLM.

RSE daemon (RSED), Lock daemon (LOCKD) and JES Job Monitor (JMON) are Developer for System z started tasks (or long-running batch jobs), each with their individual address space.

As documented in *RSE as a Java application*, RSE daemon spawns a child process for each RSE thread pool server (which supports a variable number of clients). Each thread pool is active in a separate address space (using a z/OS UNIX initiator, BPXAS).

Because these are spawned processes, they are classified via the WLM OMVS classification rules, not the started task classification rules.

The clients that are active in a thread pool can create a multitude of other address spaces, depending on the actions done by the users. Depending on the configuration of Developer for System z, some workloads, such as the TSO Commands service (TSO cmd) or CARMA, can run in different subsystems.

The address spaces listed in Figure 5 remain in the system long enough to be visible, but know that due to the way z/OS UNIX is designed, there are also several short-lived temporary address spaces. These temporary address spaces are active in the OMVS subsystem.

Note that while the RSE thread pools use the same user ID and a similar job name as the RSE daemon, all address spaces started by a thread pool are owned by the user ID of the client requesting the action. The client user ID is also used as (part of) the job name for all OMVS based address spaces stated by the thread pool.

More address spaces are created by other services that Developer for System z uses, such as File Manager (FMNCAS) or z/OS UNIX REXEC (USS build).

Classification rules

WLM uses classification rules to map work coming into the system to a service class. This classification is based upon work qualifiers. The first (mandatory) qualifier is the subsystem type that receives the work request. Table 2 lists the subsystem types that can receive Developer for System z workloads.

Subsystem type	Work description
ASCH	The work requests include all APPC transaction programs scheduled by the IBM-supplied APPC/MVS transaction scheduler, ASCH.
CICS	The work requests include all transactions processed by CICS.
JES	The work requests include all jobs that JES2 or JES3 initiates.
OMVS	The work requests include work processed in z/OS UNIX System Services forked children address spaces.
STC	The work requests include all work initiated by the START and MOUNT commands. STC also includes system component address spaces.

Table 2. WLM entry-point subsytems

Table 3 lists additional qualifiers that can be used to assign a workload to a specific service class. Refer to *MVS Planning: Workload Management* (SA22-7602) for more details on the listed work qualifiers.

		ASCH	CICS	JES	OMVS	STC
AI	Accounting Information	x		x	x	x
LU	LU Name (*)		x			
PF	Perform (*)			x		x
PRI	Priority			x		
SE	Scheduling Environment Name			x		
SSC	Subsystem Collection Name			x		
SI	Subsystem Instance (*)		x	x		
SPM	Subsystem Parameter					x
PX	Sysplex Name	x	x	x	x	x
SY	System Name (*)	x			x	x
TC	Transaction/Job Class (*)	x		x		
TN	Transaction/Job Name (*)	x	x	x	x	x
UI	User ID (*)	x	x	x	x	x

Table 3. WLM work qualifiers

Note:

For the qualifiers marked with (*), you can specify classification groups by adding a G to the type abbreviation. For example, a transaction name group would be TNG.

Setting goals

As documented in *Workload classification*, Developer for System z creates different types of workloads on your system. These different tasks communicate with each other, which implies that the actual elapsed time becomes important to avoid time-out issues for the connections between the tasks. As a result, Developer for System z tasks should be placed in high-performance service classes, or in moderate-performance service classes with a high priority.

A revision, and possibly an update, of your current WLM goals is therefore advised. This is especially true for traditional MVS shops new to time-critical OMVS workloads.

Note:

- The goal information in this section is deliberately kept at a descriptive level, because actual performance goals are very site-specific.
- To help understand the impact of a specific task on your system, terms like minimal, moderate and substantial resource usage are used. These are all relative to the total resource usage of Developer for System z itself, not the whole system.

Table 4 lists the address spaces that are used by Developer for System z. z/OS UNIX will substitute "x" in the "Task Name" column by a random 1-digit number.

Description	Task name	Workload
JES Job Monitor	JMON	STC
Lock daemon	LOCKD	STC
RSE daemon	RSED	STC
RSE thread pool	RSEDx	OMVS
ISPF Client Gateway (TSO Commands service and SCLMDT)	<userid>x	OMVS
TSO Commands service (APPC)	FEKFRSRV	ASCH
CARMA (batch)	CRA<port>	JES
CARMA (crastart)	<userid>x	OMVS
CARMA (ISPF Client Gateway)	<userid> and <userid>x	OMVS
MVS build (batch job)	*	JES
z/OS UNIX build (shell commands)	<userid>x	OMVS
z/OS UNIX shell	<userid>	OMVS
File Manager task	<userid>x	OMVS
Application Deployment Manager	CICSTS	CICS

Table 4. WLM workloads

Considerations for goal selection

The following general WLM considerations can help you to properly define the correct goal definitions for Developer for System z:

- You should base goals on what can actually be achieved, not what you want to happen. If you set goals higher than necessary, WLM moves resources from lower importance work to higher importance work which might not actually need the resources.
- Limit the amount of work assigned to the SYSTEM and SYSSTC service classes, as these classes have a higher dispatching priority than any WLM managed class. Use these classes for work that is of high importance but uses little CPU.
- Work that falls through the classification rules ends up in the SYSOTHER class, which has a discretionary goal. A discretionary goal tells WLM to just do the best it can when the system has spare resources.

When using response time goals:

- There must be a steady arrival rate of tasks (at least 10 tasks in 20 minutes) for WLM to properly manage a response time goal.
- Use average response time goals only for well controlled workloads, as a single long transaction has a big impact on the average response time and can make WLM overreact.

When using velocity goals:

- You usually cannot achieve a velocity goal above 90% because of various reasons, like that all the SYSTEM and SYSSTC address spaces have a higher dispatching priority than any velocity-type goal.
- WLM uses a minimum number of (using and delay) samples on which to base its velocity goal decisions. So the less work running in a service class, the longer it will take to collect the required number of samples and adjust the dispatching policy.
- Reevaluate velocity goals when you change your hardware. In particular, moving to fewer, faster processors requires changes to velocity goals.

STC

All Developer for System z started tasks, RSE daemon, Lock daemon and JES Job Monitor, are servicing real-time client requests.

Description	Task name	Workload
JES Job Monitor	JMON	STC
Lock daemon	LOCKD	STC
RSE daemon	RSED	STC

Table 5. WLM workloads - STC

- JES Job Monitor
  JES Job Monitor provides all JES related services like submitting jobs, browsing spool files and executing JES operator commands. You should specify a high-performance, one-period velocity goal, because the task does not report individual transactions to WLM. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be minimal to moderate.
- Lock daemon
  The lock daemon queries the GRS enqueue tables upon client and operator request, and matches the result against known Developer for System z users. You should specify a high-performance, one-period velocity goal, because the task does not report individual transactions to WLM. Resource usage is expected to be minimal.
- RSE daemon
  RSE daemon handles client logon and authentication, and manages the different RSE thread pools. You should specify a high-performance, one-period velocity goal, because the task does not report individual transactions to WLM. Resource usage is expected to be moderate, with a peak at the beginning of the workday.

OMVS

The OMVS workloads can be divided into two groups: RSE thread pools and everything else. This is because all workloads, except RSE thread pools, use the client user ID as

base for the address space name. (z/OS UNIX will substitute "x" in the "Task Name" column by a random 1-digit number.)

Description	Task name	Workload
RSE thread pool	RSEDx	OMVS
ISPF Client Gateway (TSO Commands service and SCLMDT)	<userid>x	OMVS
CARMA (crastart)	<userid>x	OMVS
CARMA (ISPF Client Gateway)	<userid> and <userid>x	OMVS
z/OS UNIX build (shell commands)	<userid>x	OMVS
z/OS UNIX shell	<userid>	OMVS
File Manager task	<userid>x	OMVS

Table 6. WLM workloads - OMVS

- RSE thread pool
  An RSE thread pool is like the heart and brain of Developer for System z. Almost all data flows through here, and the miners (user-specific threads) inside the thread pool control the actions of most other Developer for System z related tasks. You should specify a high-performance, one-period velocity goal, because the task does not report individual transactions to WLM. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be substantial.

The remaining workloads will all end up in the same service class due to a common address space naming convention. You should specify a multi-period goal for this service class. The first periods should be high-performance, percentile response time goals, while the last period should have a moderate-performance velocity goal. Some workloads, such as the ISPF Client Gateway, will report individual transactions to WLM, while others do not.

- ISPF Client Gateway
  The ISPF Client Gateway is an ISPF service invoked by Developer for System z to execute non-interactive TSO and ISPF commands. This includes explicit commands issued by the client as well as implicit commands issued by Developer for System z, like getting a PDS member list. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be minimal.
- CARMA
  CARMA is an optional Developer for System z server that is used to interact with host based Software Configuration Managers (SCMs), like CA Endevor® SCM. Developer for System z allows for different startup methods for a CARMA server, some of which become an OMVS workload. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be minimal.
- z/OS UNIX build

When a client initiates a build for a z/OS UNIX project, z/OS UNIX REXEC (or SSH) will start a task that executes a number of z/OS UNIX shell commands to perform the build. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be moderate to substantial, depending on the size of the project.

- z/OS UNIX shell
  This workload processes z/OS UNIX shell commands that are issued by the client. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be minimal.
- IBM File Manager
  Although not Developer for System z address spaces, the spawned File Manager child processes are listed here because they can be started upon request of a Developer for System z client, and these tasks use the same naming convention as Developer for System z tasks. These File Manager tasks process non-trivial MVS data set actions, like formatted editing of a VSAM file.
  Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be minimal to moderate.

JES

JES-managed batch processes are used in various manners by Developer for System z. The most common usage is for MVS builds, where a job is submitted and monitored to determine when it ends. But Developer for System z could also start a CARMA server in batch, and communicate with it using TCP/IP.

Description	Task name	Workload
CARMA (batch)	CRA<port>	JES
MVS build (batch job)	*	JES

Table 7. WLM workloads - JES

- CARMA
  CARMA is an optional Developer for System z server that is used to interact with host based Software Configuration Managers (SCMs), like CA Endevor® SCM. Developer for System z allows for different startup methods for a CARMA server, some of which become a JES workload. You should specify a high-performance, one-period velocity goal, because the task does not report individual transactions to WLM. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be minimal.
- MVS build
  When a client initiates a build for an MVS project, Developer for System z will start a batch job to perform the build. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be moderate to substantial, depending on the size of the project. Different moderate-performance goal strategies can be advisable, depending on your local circumstances.
    - You could specify a multi-period goal with a percentile response time period and a trailing velocity period. In this case, your developers should

be using mostly the same build procedure and similar sized input files to create jobs with uniform response times. There must also be a steady arrival rate of jobs (at least 10 jobs in 20 minutes) for WLM to properly manage a response time goal.

- A velocity goal is best suited for most batch-jobs, because this goal can handle highly variable execution times and arrival rates.

ASCH

In the current Developer for System z versions, the ISPF Client Gateway is used to execute non-interactive TSO and ISPF commands. Due to historical reasons, Developer for System z also supports executing these commands via an APPC transaction.

Description	Task name	Workload
TSO Commands service (APPC)	FEKFRSRV	ASCH

Table 8. WLM workloads - ASCH

- TSO Commands service
  The TSO Commands service can be started as an APPC transaction by Developer for System z to execute non-interactive TSO and ISPF commands. This includes explicit commands issued by the client as well as implicit commands issued by Developer for System z, like getting a PDS member list. You should specify a multi-period goal for this service class. For the first periods, you should specify high-performance, percentile response time goals. For the last period, you should specify a moderate-performance velocity goal. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be minimal.

CICS

Application Deployment Manager is an optional Developer for System z server that is active inside a CICS Transaction Server region.

Description	Task name	Workload
Application Deployment Manager	CICSTS	CICS

Table 9. WLM workloads - CICS

- Application Deployment Manager
  The optional Application Deployment Manager server, which is active inside a CICSTS region, allows you to securely offload selected CICSTS management tasks to developers. Resource usage depends heavily on user actions, and will therefore fluctuate, but is expected to be minimal. The type of service class you should use depends on the other transactions active in this CICS region, and is therefore not discussed in detail.

WLM supports multiple types of management that you can use for CICS:

- Managing CICS toward a region goal
  The goal is set to a service class that manages the CICS address spaces. You can only use an execution velocity goal for this service class. WLM uses the JES or STC classification rules for the address spaces but does not use the CICS subsystem classification rules for transactions.
- Managing CICS toward a transaction response time goal
  A response time goal can be set in a service class assigned to a single transaction or a group of transactions. WLM uses the JES or STC classification rules for the address spaces and the CICS subsystem classification rules for transactions.

Bibliography

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this publication.

Publication title	Order number	Reference	Reference Web site
System Programmer's Guide to: Workload Manager	SG24-6472	Redbook	http://www.redbooks.ibm.com/
ABCs of z/OS System Programming Volume 11 (performance management)	SG24-6327	Redbook	http://www.redbooks.ibm.com/
ABCs of z/OS System Programming Volume 12 (WLM)	SG24-7621	Redbook	http://www.redbooks.ibm.com/
MVS Planning: Workload Management	SA22-7602	z/OS	http://www-03.ibm.com/servers/eserver/zseries/zos/bkserv/
Resource Measurement Facility (RMF) Report Analysis	SC33-7991	z/OS	http://www-03.ibm.com/servers/eserver/zseries/zos/bkserv/

Documentation notices for IBM™ Rational™ Developer for System z

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.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 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 may not apply to you.

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

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

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

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:

Intellectual Property Dept. for Rational™ Software
IBM Corporation
3039 Cornwallis Road, PO Box 12195
Research Triangle Park, NC 27709
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 measurements 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 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.

Trademark acknowledgments

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 www.ibm.com/legal/copytrade.shtml.

Rational are trademarks of International Business Machines Corporation and Rational Software Corporation, in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, or other countries, or both.

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

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

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