

# Jopaz Documentation

Restricted Release - Not Generally Available

Version 1.2.0

March 2025

This document applies to Jopaz 1.2.0 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2022-2025 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <https://softwareag.com/licenses/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

**Document ID: JPZ-PDF-120-20250411**

# Table of Contents

<b>Jopaz Documentation.....</b>	<b>5</b>
Document Conventions.....	6
Online Information and Support.....	7
Data Protection.....	7
 <b>1 Release Notes.....</b>	 <b>9</b>
 <b>2 Getting Started with Jopaz.....</b>	 <b>11</b>
What is Jopaz?.....	12
Installation Requirements.....	12
Before You Begin Working on z/OS.....	13
Before You Begin Working on Windows.....	20
Jopaz Projects in an Integrated Development Environment.....	25
Setting up FileZilla.....	27
Configuring Jopaz.....	31
Configuring Compiler (COMOPT).....	33
Configuring Runtime (RUNOPT).....	43
 <b>3 Jopaz for Adabas.....</b>	 <b>51</b>
About Jopaz for Adabas.....	52
Installing Jopaz for Adabas.....	52
Installing Adabas Service for Java.....	53
Jopaz for Adabas Runtime Configuration.....	53
Running the Installation Verification Program PGMADAOP.class.....	55
 <b>4 Jopaz for Db2.....</b>	 <b>57</b>
Installation Parameters for Db2.....	58
Configuring Jopaz for Db2.....	59
Setting up a Db2 Password.....	61
Preparing Programs for Static Execution.....	61
Executing Applications with Db2 Access.....	67
 <b>5 Compiling with Jopaz.....</b>	 <b>69</b>
Compiling on Windows.....	70
Compiling on z/OS.....	70
Remote Debugging with Jopaz.....	79
Remote Debugging on Eclipse.....	83
 <b>6 Optimizing DFSORT.....</b>	 <b>85</b>
Runtime Settings.....	86
Faster Compiling.....	86

<b>7 Optimizing the Runtime Performance.....</b>	<b>89</b>
<b>8 Optimizing the Java Virtual Machine.....</b>	<b>91</b>
Optimizing the Java Virtual Machine.....	92
<b>9 Troubleshooting and Statistics.....</b>	<b>97</b>
Monitoring Statistics.....	99
String Index out of Bounds Exception.....	100
Activating Extended Tracing.....	101
Validating License File with LICUTIL.....	101
Increasing the Amount of Information in Error Messages.....	101
Setting the JCL TIME Parameter.....	102
Fixing Output Destination for COBOL Subprograms.....	102
Errors When Configuring Jopaz.....	102

# Jopaz Documentation

- Document Conventions ..... 6
- Online Information and Support ..... 7
- Data Protection ..... 7

---

## DISCLAIMER:

This software release is not a general availability (GA) release and is provided for restricted use only.

By using this software release, you acknowledge and accept the risks involved.

Software GmbH assumes no liability for any loss or damage resulting from its use.

Use is limited to authorized users under the terms agreed upon.

Redistribution or disclosure without prior consent is strictly prohibited.

---

This guide provides conceptual and procedural instructions for users who are working with Jopaz.

Jopaz is a COBOL application environment based on Java for IBM® Z that reduces mainframe costs without putting COBOL applications at risk.

## Document Conventions

---

Convention	Description
<b>Bold</b>	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format <i>folder.subfolder.service</i> , APIs, Java classes, methods, properties.
<i>Italic</i>	Identifies:  Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
Monospace font	Identifies:  Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the   symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

---

## Online Information and Support

---

### Product Documentation

You can find the product documentation on our documentation website at <https://documentation.softwareag.com>.

### Product Training

You can find helpful product training material on our Learning Portal at <https://learn.softwareag.com>.

### Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at <https://techcommunity.softwareag.com>. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at <https://github.com/softwareag> and <https://containers.softwareag.com/products> and discover additional Software GmbH resources.

### Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at <https://empower.softwareag.com>. Many services on this portal require that you have an account. If you do not yet have one, you can request it at <https://empower.softwareag.com/register>. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

## Data Protection

---

Software GmbH products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.





# 1 Release Notes

---

The Release Notes list the changes and enhancements introduced with Jopaz version 1.2.0.

## Jopaz Editions

Jopaz now comes in four different editions: Jopaz Lite (JPZLI), JPZ01, JPZ02, and Jopaz Bundle Batch (JPBB1). The table below shows the specifications of each edition.

Jopaz Edition	Mainframe Comiler	Runtime Compiler	Number of Applications and Main Programs	Jopaz Developer	Eclipse Plugin	Debugger	Off-Host Compiler
JPZLI	Yes	Yes	50	No	No	No	No
JPZ01	Yes	Yes	50	No	No	No	No
JPZ02	Yes	Yes	100	No	No	No	No
JPBB1	Yes	Yes	Unlimited	Yes	Yes	Yes	Yes



## 2 Getting Started with Jopaz

---

■ What is Jopaz? .....	12
■ Installation Requirements .....	12
■ Before You Begin Working on z/OS .....	13
■ Before You Begin Working on Windows .....	20
■ Configuring Jopaz .....	31

## What is Jopaz?

---

Jopaz is a powerful application environment that modernizes COBOL applications with Java to enhance productivity and control costs. Jopaz compiles custom COBOL source code into Java classes that can run on Java Virtual Machines (VMs) on IBM® zSystems® without rewriting code or retraining developers.

Since Java VMs are eligible to run on IBM zSystems Integrated Information Processor (zIIP), your modernized applications can benefit from using zIIP engines to reduce peak capacity and overall workload on the GP. With Jopaz, you can easily shift your COBOL application workloads to run on Java VMs and zIIP to save significant mainframe operating and software costs.

Other key features of Jopaz include:

- Maintaining reliable access to IBM z/OS® databases like Db2, Adabas, and VSAM without migrating to other databases or moving them off the mainframe.
- Supporting multiple development environments. Empower your teams to use Eclipse-based IBM Developer or continue to use the traditional COBOL development environment.
- Remote Debugging
- Embracing DevOps principles. Support repository-based version source control, such as GitHub, for faster, more consistent application development and delivery.
- Calling your other 3GL modules, e.g. COBOL or Assembler
- COBOL Batch support
- Sequential file support
- Print file and sorting support
- JCL Remediation
- Installation via modern z/OSMF or via JCL scripts.

## Installation Requirements

---

The Jopaz installation requirements are as follows:

- **JZOS Batch Launcher**
  - JVMJZBL1001N JZOS batch Launcher Version: 2.4.9 2021-05-27
  - JVMJZBL1002N (C) Copyright IBM Corp. 2005, 2016
  - java version "1.8.0\_301"
  - Java(TM) SE Runtime Environment (build 8.0.6.35 - pmz3180sr6fp35-20210714\_01(SR6 FP35))
  - IBM J9 VM (build 2.9, JRE 1.8.0 z/OS s390-31-Bit 20210622\_7763 (JIT disabled, AOT disabled))

## ■ z/OS

This PoC Guide is based on following z/OS Versions:

- z/OS 2.4
- z/OS 2.3

The Unix System Services (USS) always come with the same version of z/OS.

## ■ Java

We recommend that you use AdoptOpenJDK, not Oracle Java:

- IBM J9 1.8.0\_261

### Note:

You can find out your current Java version by executing `java -version` in the USS shell.

## ■ Jopaz

- JPZ-*x.x.x.x*000, where *x.x.x.x* is the Jopaz version.
- DB DynamicMVSIndexed Jopaz version 1.0
- C/S Version 82, F/S Version 19

## ■ COBOL

COBOL 4.x or newer

### Note:

If you want to use Jopaz with Adabas or Db2 databases, you must read the chapters [“Installation Parameters for Db2” on page 58](#) and [“Installing Jopaz for Adabas” on page 52](#) before you begin your Jopaz installation.

## Before You Begin Working on z/OS

### Creating a File System

To create a new file system:

1. Use one of the JCL jobs that IBM provides by default on every z/OS system disk. One of these jobs would be `JAVA.SAJVSMP1(AJVNZFS)`, for example. These TSO commands create a zSeries File System (ZFS) and format it.
2. Type in `/v sms,vol(HARDDRIVENAME),enable` to put the mainframe in a mode where the creation of a file system is possible.
3. Run the CREATE JCL job from the PoC package and adjust the cylinder size if necessary. This job also formats the file system.

## 4. Type in

```
tso mount filesystem(FILESYSTEMNAME) mountpoint('MOUNTPOINTNAME') type(zfs)
mode(rdwr)unmount
```

to make sure the file system is not moved at initial program load.

5. Type in `/v sms,vol(PLATTENNAME),D, N` to end the special mode.**You may find the following commands helpful:**

```
//pax -rwp e . /u/xxx
```

Logically copies content from one file system to another.

```
df -k .
```

Shows how big the file system is and how much is occupied.

```
tso alter NAMEFILESYSTEM newname NEWNAME
```

Renames the file system.

## Installing the Latest Java Version

To be able to install a new Java version, you must first create a new file system. For instructions on how to create one, please refer to the chapter [“Creating a File System” on page 13](#).

IBM always delivers a version of Java with each new operating system release. This is usually not the latest version. According to the new IBM convention, the Java version is located under `/global`.

1. Download Java 31 & 64 bit on your desktop PC from the IBM support site.
2. Test with a file archiver (like 7zip) if the archive is undamaged. You can also use the jarsigner to check if the .jar package is damaged. This step is optional, but recommended.
3. Upload the downloaded files via ftp (FileZilla) from the desktop to the MOUNTPOINT.

**Note:**

Upload the files as binary for software and as ASCII for text files.

4. Use `chmod 644 *.Z` and `chmod 644 *.txt` to set the permissions for the uploaded files.
5. Use `cd..` to go back to a top directory and this way unzipping the files.
6. Execute the following command:

```
pax -ppx -rvzf directoryName FolderName/SDK8_64bit_SR6_FP64.PAX.Z
```

7. Check if the folder contains the `JDK_INSTALL_OK` file. If it does, the installation ran without errors.

8. To mount in /global, it must first be removed from the old MOUNTPOINT. To mount the new one use mode read.
9. The new MOUNTPOINT should be added to SYSM.PARMLIB.

**You may find the following commands helpful:**

```
java -version
```

Displays the Java version

```
df -k .
```

Shows more details about the Java version.

**Note:**

If a .txt file is transferred to mainframe with FileZilla, then open it with Wordpad and save it again to fix formatting errors.

## Installing JZOS

JZOS consists of a load module, an example jcl, and some example programs.

If you have previously installed JZOS, it makes sense to rename the old version and keep it as backup, and use the new version as the active version.

You can always find JZOS in the Java version folder.

You have to move the the jzos tools from the directory mvstools to the mvs. Use the oget command to copy from uss to ozs mvs.

For example:

```
\u0019 tso oget'/gobal/java/j8.0/mvstools/JVMLDM80' SYS1.SIEALNKE(JVMLDM80) binary
```

## Installing Jopaz

The Jopaz package provides two ways to make the installation mostly automated.

Firstly, you can use the installation shell script jpz\_installer\_as\_shell\_script.sh. If you decide to install Jopaz this way, you must define all variables in the supplied variable input file jpz\_variable\_input\_file.txt before executing the script.

Alternatively, you can include and execute the z/OSMF workflow jpz\_installer\_workflow.xml on your z/OSMF system. For this option, the use of the provided variable input file jpz\_variable\_input\_file.txt is optional and only speeds up the installation steps.

You must choose and perform only one of the two ways to install Jopaz.

**Note:**

Before you begin your installation, please read and follow the information described in [“Before You Begin Installing Jopaz” on page 16](#).

## Before You Begin Installing Jopaz

You must do the following prerequisite steps before you begin your Jopaz installation.

1. Create an empty z/OS dataset with following attributes:

```
DSN=SAG.JPZvrs.XMIT
Dataset Organisation=PS
SPACE=(CYL,(5,5,5))
DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
```

2. Using an FTP connection in binary mode, copy the XMIT file with the same DSN as the previously created dataset and paste it on your mainframe machine.  
  
A prompt appears asking you if the existing dataset should be overwritten.
3. Confirm the prompt.
4. Copy the JCL unpack job UPJPZ.jcl using an FTP connection in ASCII mode to an accessible location on z/OS.

5. Take the supplied UPJPZ.jcl job and change:

- the location of the XMIT file. Look for SAG.JPZvrs.XMIT.
- the location of the target dataset. Look for SAG.JPZvrs.CONT.
- the *HLQ* in the RENAMEU statement.
- the CLASS and MSGCLASS parameters of the job header to their respective valid values. This depends on your machine configuration.

6. Run the job as a JES batch job.

After unpacking, there should be three datasets under the specified HLQ:

- HLQ.JPZvrs.TAR
- HLQ.JPZvrs.LOAD
- HLQ.MLCvrs.LOAD

7. If you want the MVS Load Libraries for productive use of Jopaz in a location other than the delivery High Level Qualifier, you can move them to another HLQ. Please note their new location.

**Note:**

This step is optional.

8. Find and note the location of the Jopaz license file. The required Jopaz license key is provided by Software AG logistics.



You must either use an existing license library dataset or create a new one. If you decide to create a new one, the necessary attributes for it are:

```
DSN=SAG.JPZvrs.LICS
Dataset Organisation=PO
SPACE=(CYL, (5,5,5))
DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
```

Transfer the license file to your mainframe using an FTP in ASCII mode. Place the license file (JPZBTvrs) as a member in the license library dataset.

9. Copy and uncompress the .tar.Z archive in the UNIX System Services filesystem with the following lines of code.
  - a. Create a directory in the UNIX System Services filesystem to use as the JPZ installation directory. These installation instructions use /opt/softwareag as the default installation directory, but you must change it according to your system configuration.

```
$ mkdir -p /opt/softwareag
```

- b. Copy the supplied JPZvrs.tar.Z package.

```
$ cp "'HLQ.JPZvrs.TAR(JPZTAR)'" /opt/softwareag/JPZvrs.tar.Z
```

- c. Extract the contents of JPZvrs.tar.Z.

```
$ cd /opt/softwareag
$ uncompress < JPZvrs.tar.Z | tar -oxf -
```

You should now see the jpz-installer directory.

10. Create a PDS dataset in z/OS to store the JZOS job templates of Jopaz.

You can either use the sample job zos\_create\_pds.jcl in the jpz-jcl-templates folder or create the PDS dataset in your usual way. If you decide to create the PDS dataset yourself, please make sure that the following attributes are included:

```
DISP=(NEW,CATLG,DELETE),
DSNTYPE=LIBRARY,
SPACE=(CYL,(10,10,10)),
DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
```

As an example, you can use the following notation: HLQ.JPZvrs.JOBS.

## Installing Jopaz Using the Installer Script

1. Locate the jpz\_variable\_input\_file.txt file in the /opt/softwareag/jpz-installer directory.
2. Adjust all variables in the variable input file according to your system.

Please make sure that the variable JPZ\_JOBS\_DSN= contains the exact DSN of the PDS dataset you have previously created in step 10 of [“Before You Begin Installing Jopaz” on page 16](#), the DSNs of the supplied load libraries, and DSNs of the source and copy libraries of your environment (JPZ\_SRCE\_DSN and JPZ\_COPY\_DSN).

**Note:**

During the automatic installation, a member with the name HELLO is copied into the source library (JPZ\_SRCE\_DSN). If the library you specified already has such a member, this member will be overwritten by the Jopaz installation.

When specifying the license file, enter the DSN and Member Name as two separate variables. For example, you must specify the SAG.LICENSES(JPZBT11) file as:

```
JPZ_LICENSE_DSN=SAG.LICENSES
JPZ_LICENSE_MEMBER=JPZBT11
```

3. Save the file.
4. Locate the `jpz_installer_as_shell_script.sh` in the `/opt/softwareag/jpz-installer` directory.
5. Run the script specifying the variable input file with the following command:

```
$ ./jpz_installer_as_shell_script.sh jpz_variable_input_file.txt
```

**Note:**

The variable input file might be located in another directory. In this case, enter that absolute path.

The script stops with the message `==> Setup is done.`

6. You now have the opportunity to run an automatic verification of the installation. For this, please answer with `Y`.

**Note:**

Before continuing the script, make sure that the job card is correctly defined for your system in the jobs (JPZCOMP) and (JPZEXEC) in the specified PDS.

When you continue the script execution, the verification should be completed with the message `==> Jopaz was installed successfully.`

If you do not want to run the automatic verification of the installation, end the script by entering another character. You can then run the jobs (JPZCOMP) and (JPZEXEC) manually to verify that the installation was successful.

**Note:**

If you receive an unidentified error message, please check the job spool for an error in JESMSG LG. If this is the case, you have a problem with the license file.

## Installing Jopaz Using the z/OSMF Workflow

1. Login to z/OSMF and open the **Workflows** window.
2. Click on **Activities** and select **Create Workflow**.

3. Enter the file location of the workflow:

```
/opt/softwareag/jpz-installer/jpz_installer_workflow.xml
```

4. Enter the file location of the variable input file.

You do not need to specify a variable input file, but if you define the variables in advance, it makes the installation much faster.

```
/opt/softwareag/jpz-installer/jpz_variable_input_file.txt
```

**Note:**

This step is optional.

5. Confirm your input by clicking on **Next**.
6. On the following panel, check the box **Assign all steps to own userid** and confirm with **Finish**.
7. Now you can see all steps of the workflow. Click the **+** symbol in the title column to see all substeps.
8. By clicking the title of each substep, you can see the step description. Go to the tab **Perform** to perform the step.

When the status **Complete** has been reached for all steps, the installation has been completed successfully.

**Note:**

If you receive an unidentified error message, please check the job spool for an error in JESMSG LG. If this is the case, you have a problem with the license file.

## Jopaz Folder Structure

The final structure of the Jopaz installation directory should be mapped to your USS file system as follows:

/opt/softwareag/jpz/	is the Jopaz installation root directory.
jpz-scripts/	contains the Jopaz skeleton scripts.
jpz-compiler/	is the Jopaz main folder for the compiler & runtime.
conf/	contains the configuration files for compiler & runtime.
cpy/	contains the copy books for Jopaz execution.

`err/` contains any possible error files.

`list/` contains the listing files from the Jopaz compiler.

`log/` contains any possible runtime log files.

`output/` contains the Java `.class` files.

`source/` contains your Cobol source programs.

`trace/` contains any possible tracing information.

## Before You Begin Working on Windows

---

■ Jopaz Projects in an Integrated Development Environment .....	25
■ Setting up FileZilla .....	27

## Jopaz Developer on Eclipse

For the use of Jopaz under Windows there is the Jopaz Developer for eclipse-based IDEs. In this installation guide, Software AG Designer was used as the Eclipse environment. Using the plug-in is also possible under other Eclipse environments.

### Requirements

#### 1. Unpack Archive on LINUX or Windows

Note down the location of the unpacked folder. You will need this when installing the plugin.

- On Windows:

The archive is compressed as a tar and a gzip. Two extraction steps are required.

Extract archive JPD-1.1.tar.gz to JPD-1.1.tar. Then extract archive JPD-1.1.tar to JPD-1.1. A new window will open that shows the available software.

- On Linux/Unix:

Extract the archive by using the command line `tarutility`. For example:

```
tar -xvzf JPD-1.1.tar.gz
```

#### 2. Identify the Jopaz Developer License

This will be provided separately from Software AG and is specialized for the operating system. The name of the license file is fixed: `jopaz-developer.xml`. The location of the license file is relevant for the following installation, so note its location.

### Installing Jopaz

#### 1. You must specify the license file location. This happens in two ways:

- **User Home directory (Default)**

If you have not specified an environment variable, the user home directory is set as the default location of the license file.

- **Using Environment Variable `JOPAZ_LICENSE`**

Set the path to the license file with the environment variable `JOPAZ_LICENSE`.

**Note:**

The name of license file is fixed: `jopaz-developer.xml`.

#### 2. Add archive to Eclipse. Go to menu **Help > Install new Software**.

A new window opens showing the available software.

3. Click on **Add**.
4. Fill in the required information and then click on **Add**.

- **Name:** Jopaz

- **Location:** file:/path-of-JPD-1.1

where *path-of-JPD-1.1* is the path of the Jopaz Developer.

The new repository is selected automatically after adding.

5. Select the packages to install. Click on **Next** to start the installation of the Jopaz Developer plugin.
6. Follow the steps of the installation.

During the installation a popup message appears and warns that software from a noncertified or unsafe location is to be installed. Accept and continue the installation.

**Note:**

This plugin is useful to get a first impression of the functionality. But it only can be used on Windows or Linux machines. This functionality is not available for z/OS or USS.

### Opening a Jopaz Perspective

After installing the compiler plugin, a new perspective is available in Eclipse.

1. Open the perspective by selecting **Window > Perspective > Open Perspective > Other**.
2. Select **Jopaz** and click **Open**.

The perspective has opened if you can see **Jopaz Tools** in the navigation bar.

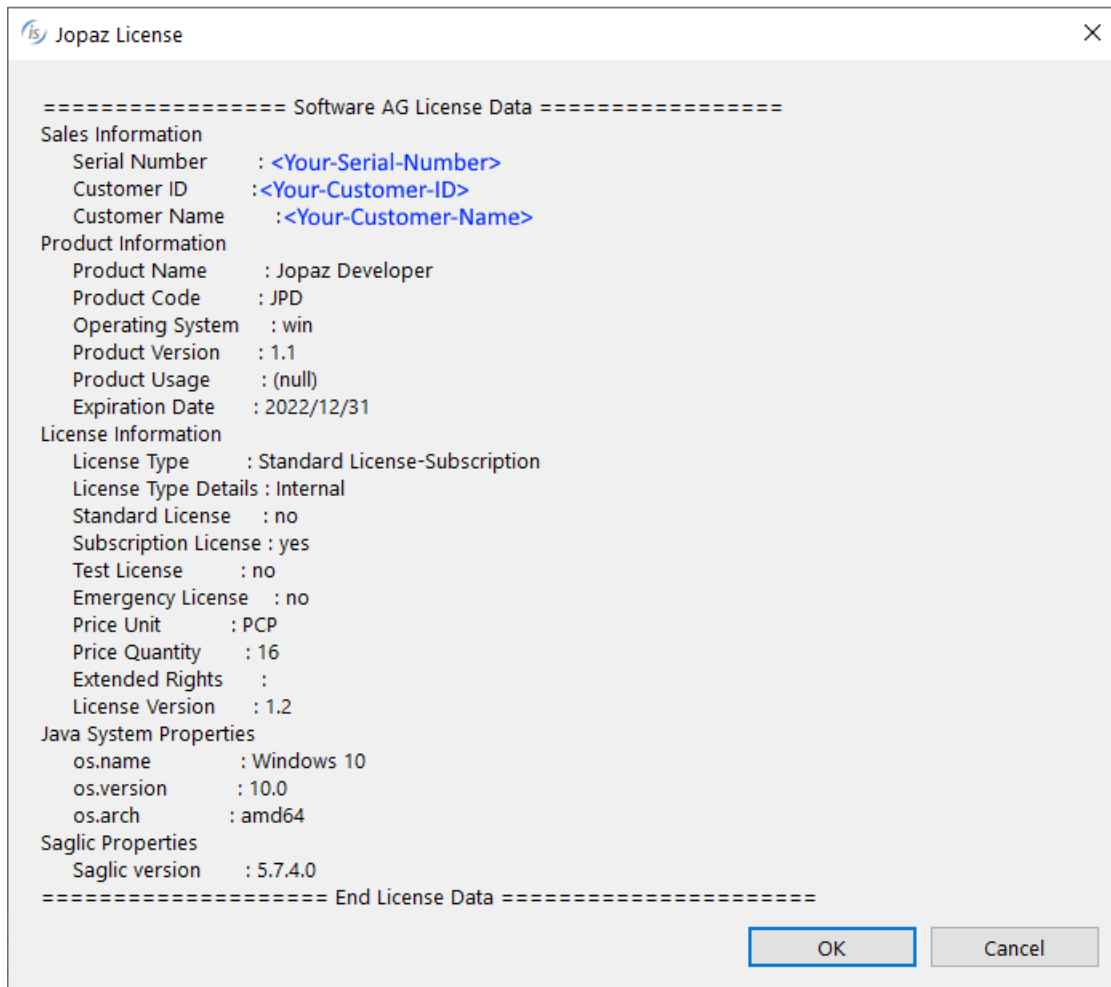
### Checking License Status

You must open the Jopaz perspective when adding or updating Jopaz licenses.

1. Select **Help > Jopaz License Status**.

The **Jopaz License Settings** panel opens.

2. The license information should look like this:



3. Close the license file by clicking on **OK**.

## Adding Java (JRE/JDK) for Jopaz

You set this option only once.

1. Go to **Window > Preferences**.
2. From the menu, select **Jopaz**.
3. Set the java compiler executable by clicking on **Browse** and selecting the required javac.exe file.





# Jopaz Projects in an Integrated Development Environment

## Creating a Project

1. Open **File > New > Jopaz Project**.
2. Select the project of your choice and click **Finish**.

Switch to Jopaz perspective and open the view **File** to have a better view of the Jopaz project structure.

## Configuring Project Settings

1. Right click on **Project** and select **Properties > Jopaz Settings > Compile/Runtime**.
2. Set **Current Options** to the options described in section [“Log4j Vulnerability Optimization” on page 92](#) located in Optimizing the Java Virtual Machine.

## Compiling and Running COBOL sources in Eclipse

1. Right click on a source and click **Compile**.

Depending on the compiler settings a java .class file will be created in the output folder.

You have the option to select multiple sources as well.

2. To run the program, right click on the source and select **Run As > Jopaz Application**.



# Setting up FileZilla

## Installing FileZilla

You can download FileZilla Client form the FileZilla website.

## Setting up Secure Connection

### Setting up Secured FTP (FTPS) on z/OS

1. Open FileZilla.
2. Click on **File > Site Manager**.
3. Click on **New site**.
4. Enter a session name.
5. On the **General** tab select the following:
  - **Protocol:** FTP – File Transfer Protocol
  - **Host:** *Your personal hostname*
  - **Encryption:** Require explicit FTP over TLS
  - **Logon Type:** Ask for password
6. On the **Advanced** tab:
  - Set the **Server type:** MVS, OS/390, z/OS.
  - Set a path under **Default local directory**.
  - Enter a path under **Default remote directory**.
7. Click on **Connect**.
8. Enter your userID and you mainframe password.
9. Click **OK**.

You will be presented with a server certificate overview window. Click **OK** and your session is now established.

### Setting up Secured FTP (FTPS) on USS (Open Edition)

1. Select **File > Site manager**.
2. Click on **New site**.
3. On the **General** tab select the following:
  - **Protocol:** FTP – File Transfer Protocol
  - **Host:** *Your personal hostname*
  - **Encryption:** Require explicit FTP over TLS
  - **Logon Type:** Ask for password
4. On the **Advanced** tab:
  - Set the **Server type:** Default (Autodetect).
  - Set a path under **Default local directory**.
  - Enter your USS home directory for **Default remote directory**.
5. Click on **Connect**.
6. Enter your userID and you mainframe password.
7. Click **OK**.

You will be presented with a server certificate overview window. Click **OK** and your session is now established.

### Setting up Secured FTP with SSH (SFTP)

1. Select **File > Site manager**.
2. Click on **New site**.
3. On the **General** tab select the following:
  - **Protocol:** SFTP – SSH File Transfer Protocol
  - **Host:** Enter hostname
  - **Encryption:** Require explicit FTP over TLS
  - **Logon Type:** Ask for password
4. On the **Advanced** tab:

- Set the **Server type**: Unix.
  - Set a path under **Default local directory**.
  - Enter your USS home directory for **Default remote directory**.
5. Click on **Connect**.
  6. Enter your userID and you mainframe password.
  7. Click **OK**.

You will be presented with a server certificate overview window. Click **OK** and your session is now established.

## Setting File Transfer Types

In order to set File Transfer Types for:

- **Auto** Recognizing:
  1. Select **Edit > Settings... > Transfers > FTP: File Types**.
  2. Add missing file types to list for automatic file type transition. A dot is not required.
  3. Click **OK**.
- **Manual** Recognizing:
  1. Navigate to **Transfer > Transfer Type** and select either Auto, ASCII, or Binary.

## Setting up MobaXTerm

### Note:

You can also use MobaXTerm as an SSH client, but you will need a license.

## Creating an SSH session

1. Open MobaXTerm.
2. Click on the tab **Sessions** at the top.
3. Select **New Session**.
4. Choose **SSH**.
5. Configure SSH session accordingly.

6. Set hostname.
7. Click on **OK**.

## Opening an SSH Session

You open an SSH Session to connect to USS.

1. Start SSH session by double clicking the session in the left panel.
2. Login with your mainframe credentials.

# Configuring Jopaz

---

- [Configuring Compiler \(COMOPT\) .....](#) 33
- [Configuring Runtime \(RUNOPT\) .....](#) 43





# Configuring Compiler (COMOPT)

## Default Configuration

This section lists the options defined by default with an explanation of each compiler option.

```
# COMPILER SETTINGS
# There must be TWO SPACES before a line break "\"
jopaz.compiler.options=-b -cod1 -crv -la \
-od=<baseDir>/output
```

In this example *JPZ-INSTALLATIONPATH* corresponds to the location of your Jopaz project and *path-to-trace* is the location of your logfile.

### Note:

The paths you define here must be consistent with the paths you specify in the templates from chapter [“Compiling and Running with Jopaz using JZOS” on page 71](#).

## Specifying the MVS Source File Location

You can specify the location of the source file as a DSN DD card in the compile job instead of specifying a UNIX System Services (USS) directory in the classpath where the source file is located. This eliminates the need to transfer existing sources from a dataset to a USS directory.

However, if the COBOL sources are to be stored in the USS, this option must be switched off manually:

```
# USING MVS PATHS IN COMPILE JCL TO SPECIFY THE SOURCE FILE
# - 1: SPECIFY SOURCE MEMBER IN 'JPZSRC' DD CARD
# - 0: SOURCE FILE IS TAKEN FROM 'SOURCE' DIRECTORY
jopaz.compiler.mvs_source=0
```

For more information, see the description of this property in [“Compiler Property Definition” on page 35](#).

## Specifying the MVS Copybook Location

You can specify the location of the copybooks as a DSN DD card in the compile job instead of specifying a UNIX System Services (USS) directory in the compiler option `-sp` where the copybooks are located. This eliminates the need to transfer existing copybooks from a dataset to a USS directory.

However, if the COBOL copybooks are to be stored in the USS, this option must be switched off manually:

```
# USING MVS PATHS IN COMPILE JCL TO SPECIFY THE COPYBOOKS LOCATION
# - 1: SPECIFY LOCATION OF COPYBOOKS IN 'SYSLIB' DD CARD
# - 0: SPECIFY LOCATION OF COPYBOOKS IN -sp COMPILER OPTION
jopaz.compiler.mvs_syslib=0
```

For more information, see the description of this property in [“Compiler Property Definition” on page 35](#).

## Warnings

You can stop receiving particular warnings by using the following option. You can list more than one warning.

The following example shows how to suppress the warning 194:

```
type conf.comp
jopaz.compiler.messagelevel.194=0
```

## Debugging Option

To use debugging in Jopaz, you must set the options `-cv` and `-d` and you must set the framework property for debugging `jopaz.use` accordingly.

```
# DEBUGGING USAGE
jopaz.use_for_debugging=1
```

## Setting the Compiler Listing Output Location

You can set three different options for the output of the compiler listing. The first activates the writing of the listing into the job class `STDOUT` when the compile job is executed. The second activates the writing of the listing into a UNIX file under the path specified in the compiler option `-lo`. The third combines option one and two. The default value is the `STDOUT` job class.

```
# LOCATION OF COMPILER LISTING
# 1: STDOUT (DEFAULT)
# 2: USS FILE - PATH OF -lo COMPILER OPTION
# 3: BOTH (MVS + USS)
jopaz.list_location=1
```

## Improving the support for ESQL on IBM Db2

The additional compiler property `jopaz.compiler.esql.db2` signals the Jopaz compiler that the database used in the application is IBM Db2. The property is set to "1" (true) by default.

If you do not disable this property, the compiler generates specific code, which returns the result sets in the same format as the IBM Db2 precompiler. It supports the `SQLDA` structure and the use of date, time, and timestamp as function parameters.

```
jopaz.compiler.esql.db2=1
```

## Dealing with Reserved Words in the Cobol Source

Sometimes you might have words in your Cobol source that are considered reserved by Jopaz. Jopaz might misinterpret such words and you can receive compiler errors such as #144 `Invalid name` and #15 `Unexpected token`. You can specify those words with the `-rw` compiler property to avoid receiving errors.

The following example uses the reserved words `POS` and `CONVERSION`.

```
jopaz.compiler.options= ... -rw=POS,CONVERSION
```

## IBM Hexadecimal Floating-Point Format

If you want the data item to be stored using IBM's float or double hexadecimal floating-point format, add the following runtime property to your properties file.

This also applies to COMP-1 or COMP-2 using the compiler option `-cv`.

```
jopaz.floating_point_format=ibm_hfp
```

## Compiler Property Definition

Option	Description
<code>jopaz.compiler.adaname=literal</code>	<p>This property informs the compiler if an alternative name other than CALL 'ADABAS' is used for the Adabas call in the COBOL code. The alternative call name is then translated during compilation so that calls for Jopaz for Adabas are executed without errors.</p> <p>The value of this property must be equal to the exact name of the alternative call.</p>
<code>jopaz.compiler.command_line_linkage=</code>	<p>This property is necessary if parameters are to be passed to the linkage section of the application when translating using the ARGS statement in the JCL.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – The command line parameters are passed to the linkage section.</li> <li>■ 0 – The command line parameters are not included in the linkage section.</li> </ul> <p>The default value is 1.</p> <p>The command line is passed to the main program using the following parameter structure:</p> <pre>LINKAGE SECTION. 01 CmdLine.   03 CmdLineLen pic 9(4)comp.   03 CmdLineData pic x(256).</pre> <p>In the JCL call to run the compiled application, the parameters must be passed in the following way:</p> <pre>//JVMEXEC EXEC PROC=JVMPRC86,REGSIZE=512M,LOGLVL='', // JAVACLS='PGMHCL','',ARGS='TestParameter'</pre>
<code>jopaz.use_for_debugging=</code>	<p>This property controls the USE FOR DEBUGGING declarative, which shows how many times a routine runs.</p>

Option	Description
	<p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – Enable the <code>USE FOR DEBUGGING</code> declarative for programs compiled with the <code>-cv</code> option.</li> <li>■ 0 – The <code>USE FOR DEBUGGING</code> declarative is disabled.</li> </ul> <p>The default value is 0.</p> <p>This option has a big impact on performance, so it is recommended to be enabled only if you plan to debug the application you are compiling with certainty.</p> <p><b>Note:</b> You must set the options <code>-d</code> or <code>-dx</code> when enabling this debugging option.</p>
jopaz.compiler.mvs_source=	<p>This property allows you to directly specify a physical sequential dataset or a dataset member as a source for the compilation.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – Enable the source specification using the DSN.</li> <li>■ 0 – Disable the source specification using the DSN.</li> </ul> <p>The default value is 1.</p> <p>You must specify the complete dataset name in the DD card with the name <code>JPZSRC</code>. For example:</p> <pre>//JPZSRC DD DISP=SHR,DSN=HLQ.MY.DATASET(PGMHEL)</pre> <p>If you set the value of this property to 0, you need to specify the full path of the source in a Unix System Services directory in the <code>ARGS</code> statement of the JCL.</p>
jopaz.compiler.mvs_syslib=	<p>This property allows you to specify a dataset directly as the copybook library for the compilation.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – Enable the specification of copybook library using the DSN.</li> <li>■ 0 – Disable the specification of copybook library using the DSN</li> </ul> <p>The default value is 1.</p> <p>The complete dataset name must be specified in the DD card with the name <code>SYSLIB</code>. If you want to use a directory from the Unix</p>

Option	Description
	System Services as a copybook library, this property must be set to 0 and the path to the directory must be specified using the compiler option -sp.
jopaz.list_location=	<p>This property determines the location where the compiler listing is generated or output.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>1 – The listing is printed to the STDOUT job class.</li> <li>2 – A listing file with the name of the source is placed in the Unix System Services directory, which can be determined with the compiler option -lo.</li> <li>3 – Both options are used (STDOUT and UNIX directory).</li> </ul> <p>The default value is 1.</p>
jopaz.floating_point_format=	<p>This property specifies the format used to store Float and Double values.</p> <p>Under -cv compiler option this property affects also COMP-1 and COMP-2 data items, as they are treated as Float and Double respectively.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ ibm hfp</li> <li>■ ieee 754</li> </ul> <p>The default value is ibm hfp.</p>
jopaz.compiler.esql.db2=	<p>This compiler property informs the compiler that the database used in the application is IBM Db2.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>1 – The compiler generates specific code to return the result sets in the same format as the IBM DB2 pre-compiler. In particular, it supports the SQLDA structure and the use of date, time, and timestamp as function parameters.</li> <li>0 – No IBM Db2 specific rules are applied during compilation.</li> </ul> <p>The default value is 1.</p>

## Compiler Options

Option	Description
-apost	Causes figurative constants QUOTE and QUOTES to be considered as single quotation marks.
-b	Treats characters as bytes in STRING, UNSTRING, and INSPECT statements.
-cax	Specifies the default file assignment as external.
-cod1	<p>Affects data items that appear after a variable-length table in the same record.</p> <p>With -cod1, these items always immediately follow the table, regardless of its size.</p> <p>The addresses of these items changes as the size of the tables changes.</p>
-coe	Closes all files opened by the program at program exit.
-crv	Sets a varying size of implicit records for files with multiple record definitions with different lengths, and for files with varying records. If you do not set -crv, files are treated as fixed length and the maximum record length is used. This option affects files that include two or more record definitions with different sizes as well as files that include -cod1.
-csl	Treats the COBOL name in ASSIGN clause as a literal.
-cudc	<p>Treats numeric USAGE DISPLAY data as characters during a process of comparison or moving. If you set this option, numbers with a DISPLAY usage are compared byte by byte instead of comparing their numeric representation.</p> <p>The byte by byte comparison is used:</p> <ul style="list-style-type: none"> <li>■ when comparing two unsigned numbers with DISPLAY usage having the same length and the same number of decimal digits.</li> <li>■ when comparing an unsigned integer number and an alphanumeric elementary item with the same length.</li> <li>■ an unsigned number with DISPLAY usage with ZERO (ZEROES, ZEROS) or 0.</li> </ul> <p>The comparison is made comparing each digit within the number, byte by byte, with the character 0.</p> <ul style="list-style-type: none"> <li>■ when comparing an unsigned number with DISPLAY usage with SPACE (SPACES) or "".</li> </ul>

Option	Description
	<p>The comparison is made comparing each digit within the number, byte by byte, with the character " ".</p> <p>This option affects the MOVE statement when one operand is a USAGE DISPLAY unsigned numeric data item, a numeric constant, or a numeric literal and the other one is an unedited alphanumeric item.</p> <ul style="list-style-type: none"> <li>when the sender operand is an alphanumeric data item and the receiver operand is a USAGE DISPLAY numeric data item</li> </ul> <p>During the byte by byte move, the sender operand is considered as containing only one digit, the string representation of an integer number, and no check is performed on the real content.</p> <p>For example:</p> <pre>MOVE "FA" TO PIC-XX MOVE PIC-XX TO PIC-9 = A MOVE PIC-XX TO PIC-99 = FA MOVE PIC-XX TO PIC-999 = 0FA MOVE PIC-XX TO PIC-Z = A MOVE PIC-XX TO PIC-ZZ = FA MOVE PIC-XX TO PIC-ZZZ = 0FA MOVE PIC-XX TO PIC-V9 = 0 MOVE PIC-XX TO PIC-9V9 = A0 MOVE PIC-XX TO PIC-9V99 = A00</pre> <ul style="list-style-type: none"> <li>when the sender operand is a USAGE DISPLAY unsigned numeric data item and the receiving operand is an unedited alphanumeric data item.</li> </ul> <p>During the byte by byte move, the first operand is considered as an alphanumeric item.</p> <ul style="list-style-type: none"> <li>when the sender and the receiver operands have an identical PICTURE and USAGE.</li> </ul> <p>During the byte by byte move, both operands are considered as alphanumerics.</p>
-cv	<p>IBM COBOL compatibility flag. This flag ensures that the Jopaz compiler supports IBM COBOL specific syntax. For example: USE FOR DEBUGGING and WITH DEBUGGING MODE.</p>
-cva	<p>IBM arithmetic compatibility.</p> <p>This flag ensures that IBM COBOL specific arithmetic operations are performed correctly.</p> <p>In the example compute <math>res = (11/4) * 4</math></p>

Option	Description
	<ul style="list-style-type: none"> <li>■ If you declare <code>res</code> as PIC 99, <code>res</code> is set to 8.</li> <li>■ If you declare <code>res</code> as PIC 99v99, <code>res</code> is set to 11.00.</li> <li>■ If the <code>-cva</code> option is not used, the result is 11.</li> </ul>
<code>-d</code>	<p>Includes debug information. An additional class file is generated to store debug information. The resulting classes don't include the source code.</p> <p>Source files must be available to the Debugger during the debug session.</p>
<code>-dx</code>	<p>Enables extended debugger functions. This option implies <code>-d</code>.</p> <p>All variables in the class are generated. All variables that are not used in the program are also generated. The literal constants that are generated during the execution are generated not as static fields in the generated class.</p> <p>The Debugger is able to query and set all the items of the program Data Division including the items that are not used in the Procedure Division. The IDE allows the source code to be changed while debugging.</p>
<code>-dcii</code>	<p>Uses IBM sign encoding and IBM COMP sizes.</p> <p>COMP sizes are 2, 4, 8, or 16 depending on the item picture.</p>
<code>-dv=0</code>	<p>If <code>-dv</code> is omitted, the compiler behaves as if <code>-dv=32</code> was specified. That means that data items specified without a <code>VALUE</code> clause are filled with ASCII spaces by default.</p>
<code>-dznt</code>	<p>This option represents the IBM COBOL compiler option <code>TRUNC(STD)</code>.</p> <p>This flag provides a relaxation of the size checking rules in compatibility with the micro focus <code>NOTRUNC</code> directive.</p> <p>If this flag is set, the values that can be held in binary data types are limited only by the number of bytes of memory.</p>
<code>-la</code>	<p>Outputs full listing in ANSI fixed format. The list file contains all the source code. All the copybooks are merged into it unless the <code>SUPPRESS</code> clause is used in the <code>COPY</code> statement. In most cases, it can be compiled as this is a standard COBOL program.</p> <p>If used along with <code>-ld</code>, only the source part is generated in ANSI fixed format, datamaps are always in free format.</p>



Option	Description
-ld	<p>Outputs full listing and data map to a .list file. The file contains all of the source code and the datamaps. All copybooks are merged into it unless the SUPPRESS clause is used in the COPY statement.</p> <p>The datamap information is stored at the bottom of the list file and provides the following information for each data item described in the program Data Division: source line, item name, offset in the case that the item is part of a group item, physical length, section in which the item is defined, type flags, item type, and how the item is referenced in the Procedure Division.</p>
-lf	Outputs full listing to a .list file.
-lo=DirName	Specifies the directory where .list files are to be stored.
-m1	<p>Puts all of WORKING-STORAGE into a contiguous block of memory.</p> <p>If you use the -m1 option, you must comment out or remove the property <code>jopaz.array_check=1</code>, or change the value to 0.</p>
-noadv	<p>This option is used as an IBM COBOL compatibility for compiler option NOADV.</p> <p>Prevents the file from being created with an additional byte in the Logical Record Length (LRECL) if you are using a record format (RECFM) with ANSI (A) or machine code (M) formatting.</p> <p>Use -noadv if you already adjusted record length to include 1 byte for the printer control character.</p>
-od=DirName	Specifies the output directory for classes.
-pt2	<p>The -pt options control the behavior of returns from code executed during a PERFORM statement.</p> <p>The flag -pt2 supports compatibility with mainframe behavior of OS/VS COBOL, DOS/VS COBOL, VS COBOL II, and COBOL/370.</p>
-rw=word	Suppresses reserved words. Multiple words must be separated by commas. For example, when a program contains variable 77 PRINTER PIC X(32), this option must be used: -rw=PRINTER
-sa	<p>Specifies Force Fixed (also known as ANSI) source format. The same effect can be obtained using the SOURCE directive by specifying at the top of the source file:</p> <pre>&gt;&gt; SOURCE FORMAT FIXED</pre>
-sp=Copypath	Specifies all paths in which COPY files can be found.

Option	Description
-verbose	Displays verbose output such as the count of errors, informational, and warnings.
-wdbz	Show warnings for possible division by zero without ON SIZE ERROR messages.
-xjc	Generates the .class file.

## Notes

- You can set compiler options both in the source code and in the general configuration in COMOPT via the `jopaz.compiler.options=` property. Setting compiler options this way works similarly to other COBOL compilers with inline compiler options.
- The `IMP OPTION` directive sets compiler options for a program. This directive must appear as first row in the source file and must start from column 12. It cannot be used in the body of the source code. For example:

```
>>IMP OPTION "-noadv"  
IDENTIFICATION DIVISION.  
PROGRAM-ID. PRTFCDEX.
```

# Configuring Runtime (RUNOPT)

## External DFSORT

To use the external sort algorithm DFSORT, include following lines in RUNOPT:

```
# DFSORT SETTINGS
jopaz.sort=com.softwareag.jopaz.sort.SAGSort
```

To use DFSORT, you must specify the following libraries in the STEPLIB DD card:

```
//STEPLIB DD DISP=SHR,DSN=SYS1.SICELPA
//          DD DISP=SHR,DSN=SYS1.SICELINK
//          DD DISP=SHR,DSN=SYS1.SORTLPA
//          DD DISP=SHR,DSN=SYS1.SORTLIB
```

## Runtime Debugging

To be able to debug the Jopaz application remotely, you must set the following properties:

```
jopaz.rundebbug=2
jopaz.debug.port=9999
```

## DB2 Declaration

To define the connection and configuration of DB2 for Jopaz there are two options.

### Option 1

In the first option, the parameters are set directly in the RUNOPT file:

```
jopaz.jdbc.url=jdbc:db2://db2_host_or_IP:DB2_port/database
jopaz.jdbc.options=user=DB2_user,password=DB2_password
jopaz.jdbc.driver=com.ibm.db2.jcc.DB2Driver
jopaz.jdbc.auto_connect=1
jopaz.jdbc.dateformat=Date_format_e.g._dd.MM.yyyy
jopaz.jdbc.kept_spaces=2
jopaz.jdbc.options=fixedString=false
```

where *db2\_host\_or\_IP* is your IP or host number, *DB2\_port* is the number of the DB2 port, *database* is the name of your database, and *DB2\_user* and *DB2\_password* are your personalized username and password.

### Option 2

The second option uses a separate data source class:

```
jopaz.jdbc.datasource=DataSource
jopaz.jdbc.driver=com.ibm.db2.jcc.DB2Driver
jopaz.jdbc.auto_connect=1
jopaz.jdbc.dateformat=Date_format_e.g._dd.MM.yyyy
jopaz.jdbc.kept_spaces=2
jopaz.jdbc.options=fixedString=false
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import com.jopaz.rts.MyDataSource;
public class DataSource implements MyDataSource {
    private static final String URL = "jdbc:db2://DB2_host_or_ip:DB2_port/Database";
    private static final String USER = "DB2_user";
    private static final String PASSWORD = "DB2_password";
    @Override
    public Connection connect(String db, String user, String password) throws
SQLException {
        String connUser = user != null && !user.trim().equals("") ? user : USER;
        String connPassword = password != null && !password.trim().equals("") ?
password : PASSWORD;

        try {
            Class.forName("com.ibm.db2.jcc.DB2Driver");
        } catch (ClassNotFoundException e) {
            throw new SQLException(e);
        }

        return DriverManager.getConnection(URL, connUser, connPassword);
    }
}
```

where *Package\_name* is the name of the package you are using, *db2\_host\_or\_IP* is your IP or host number, *DB2\_port* is the number of the DB2 port, *database* is the name of your database, and *DB2\_user* and *DB2\_password* are your personalized username and password.

## ADS Dumps on Error

You can obtain an ADS (Abend Diagnostic Snapshot) dumps in the event of an error by setting the following properties. The recommended value for the `jopaz.exception.message` and `jopaz.display.message` options is 2, so all messages are written directly to `syserr`.

```
jopaz.exception.dump=1
jopaz.exception.message=2
jopaz.display.message=2
# THIS OPTION IS ONLY NECESSARY IF YOU HAVE SET THE VALUE 3 IN
# jopaz.exception.message AND/OR jopaz.display.message
jopaz.exception.dumpfile=/tmp/%p.dump
```

## Runtime Property Definition

Option	Description
<code>jopaz.array_check=</code>	<p>Checks array boudaries.</p> <p>Valid values are:</p> <ul style="list-style-type: none"><li>■ -1 – Array boundaries are checked at Runtime in order to provide more details in case "out of bounds" errors exist. When a program addresses an item that is outside the valid range, an error message is written to the log if you have set <code>jopaz.tracelevel</code> to a value</li></ul>

Option	Description
	<p>greater than zero. The error message informs about the data item name and the problematic index value.</p> <ul style="list-style-type: none"> <li>■ 0 – Array boundaries are not checked. If a program addresses an item that is outside the valid range, a generic "out of bounds" error message is shown and the program exits. The <code>-m1</code> option may avoid the crash and force the program to access the area of the next Working-Storage item instead.</li> <li>■ 1 – Array boundaries are checked at Runtime in order to provide more details in case "out of bounds" errors exist. If a program addresses an item that is outside the valid range, an error message is shown and the program exits. The error message informs about the data item name and the problematic index value.</li> </ul> <p>The default value is 0.</p>
jopaz.checkdiv=	<p>Allows you to specify an alternate runtime response to a divide by zero condition when the statement does not include a <code>ON SIZE ERROR</code> clause.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ -1 – The error message "Attempt to divide by zero" is written to the log and the program aborts. <code>jopaz.tracelevel</code> must be set to a value greater than zero.</li> <li>■ 0 – Results are undefined.</li> <li>■ 1 – The program aborts with the error message "Attempt to divide by zero".</li> <li>■ 2 – The result is zero.</li> <li>■ 3 – The result is the dividend, as if the division was by 1 instead of by zero.</li> </ul> <p>The default value is 0.</p>
jopaz.cpu_statistics=	<p>Enables the Jopaz CPU statistics for runtime.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – Enable the tracing for CPU statistics,</li> <li>■ 0 – Disable the tracing for CPU statistics.</li> </ul> <p>With the DD card <code>JPZPRINT</code> the CPU statistics can be written either to a separate job class or to a physical file. Otherwise the statistics are written in the JES Job Log (JESMSGLG).</p>

Option	Description
jopaz.debug.port=	<p>This property specifies the port used by the Remote Debugger. It can be used in the Framework configuration to tell which port it should listen to connect instead of passing this information on the command line.</p> <p>The default value is 9999.</p>
jopaz.display_message=	<p>This property defines how messages are shown to the user. It affects both messages displayed by the program through the <code>DISPLAY MESSAGE BOX</code> statement and error messages shown by the runtime system. In thin client environments, it is also evaluated on the client side for the client connection error messages.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 0 – All messages are shown in a message box.</li> <li>■ 1 – All messages are sent to sysout. In a thin client environment, runtime error messages are sent to the server sysout, while messages displayed by the program are sent to the client sysout.</li> <li>■ 2 – All messages are sent to syserr. In a thin client environment, runtime error messages are sent to the server syserr while messages displayed by the program are sent to the client syserr.</li> <li>■ 3 – Messages displayed by the program are sent to syserr or clientsyserr in thin client environments. Runtime error messages are printed to a file named <code>&lt;program_name&gt;&lt;number&gt;.ads.log</code>, where <code>&lt;number&gt;</code> is a progressive number calculated by the runtime. The file name and location can be customized with the <code>jopaz.exception.dumpfile</code> property.</li> </ul> <p>Any other value is equivalent to 2. The default value is 0. The recommended value is 2.</p>
jopaz.exception.dump=	<p>This property defines wheather to produce an Abend Diagnostic Snapshot (ADS) or not.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – Produces ADS in addition to Java exceptions.</li> <li>■ 0 – Does not produce ADS.</li> </ul> <p>The default value is 0.</p>
jopaz.exception.dumpfile=pathname	<p>This property specifies the pathname of the file generated when setting <code>jopaz.display_message=3</code> or <code>jopaz.exception.message=3</code>. The following special characters are supported in the value of this property:</p>

Option	Description
	<ul style="list-style-type: none"> <li>■ %p – program name</li> <li>■ %d – current date in the form YYYYMMDD</li> <li>■ %t – current time in the form HHMMSSSTTT</li> <li>■ %u – username</li> <li>■ %h – hostname.</li> </ul> <p>If the value begins with a "+" character, the report is appended to the specified file, otherwise the new report overwrites the specified file. For example:</p> <pre>jopaz.exception.dumpfile=/tmp/%p.dump</pre> <p>If this property is not set, a file named <code>&lt;program_name&gt;&lt;number&gt;.ads.log</code>, where <code>&lt;number&gt;</code> is a progressive number calculated by the runtime, is generated by default in the working directory.</p>
jopaz.exception.java=	<p>Specifies the content of the exception messages.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – Internal java methods are traced in exception messages.</li> <li>■ 0 – COBOL paragraph names are traced in exception messages.</li> </ul> <p>The default value is 0.</p>
jopaz.exception.message=	<p>This property defines how exception messages are shown to the user. It affects only error messages shown by the runtime system.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 0 – The messages are shown in a message box.</li> <li>■ 1 – The messages are sent to sysout. In a thin client environment, they are sent to the server sysout.</li> <li>■ 2 – The messages are sent to syserr. In a thin client environment, they're sent to the server syserr.</li> <li>■ 3 – The messages are printed to a file named <code>&lt;program_name&gt;&lt;number&gt;.ads.log</code>, where <code>&lt;number&gt;</code> is a progressive number calculated by the runtime. The file name and location can be customized with the <code>jopaz.exception.dumpfile</code> property.</li> </ul> <p>Any other value is equivalent to 2. The default value is 0. The recommended value is 2.</p>

Option	Description
<code>jopaz.file.close_on_exit=</code>	<p>Specifies open files to close at program exit.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – All open files are closed when the program exits.</li> <li>■ 0 – Open files are left open when the program exits.</li> </ul> <p>The default value is 0.</p>
<code>jopaz.file.errors_ok=</code>	<p>Configures how I/O errors are treated.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 0 – The program stops if an error occurs and there are no declaratives.</li> <li>■ 1 – The program continues even if an errors occurs.</li> <li>■ 2 – The program continues when an error occurs only if a status is defined for the file.</li> </ul> <p>The default value is 0.</p>
<code>jopaz.gui.show_zeroes=</code>	<p>Specifies whether leading zeros are shown when displaying numeric data.</p> <p>This property affects all the output devices: console, character-based terminals, and graphical windows.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – Leading zeroes are shown when displaying numeric data.</li> <li>■ 0 – Leading zeroes are not shown when displaying numeric data</li> </ul> <p>The default value is 0.</p>
<code>jopaz.logfile=pathname</code>	<p>This property specifies the path of the log file. Backslashes must be doubled.</p> <p>The Jopaz framework uses the <code>java.util.logging</code> package and there are many configuration options.</p> <p>For example, you can specify <code>%h</code> in the <code>jopaz.logfile</code> and it will be replaced by the user's home directory.</p> <p>You can specify <code>%yyyy</code>, <code>%mm</code>, <code>%dd</code>, <code>%hh</code>, <code>%nn</code>, <code>%ss</code>, and <code>%cc</code> and they will be replaced with the current year, month, day, hour, minute, second, and hundredth of second respectively.</p> <p>You can specify a <code>%u</code> in the <code>iscobol.logfile</code> and it will be replaced with a unique number at runtime to resolve conflicts.</p>



Option	Description
jopaz.rundebbug=	<p>This property specifies how the Runtime interacts with the remote debuggers.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 0 – Remote debugging is not possible.</li> <li>■ 1 – The first program will start and run immediately. The remote debug session waits and listens for a program that is compiled for debugging to launch connecting to the program when it detects it. This is useful when you want to debug only some of your application's programs.</li> <li>■ 2 – The runtime framework will start in debug mode pausing to connect to a remote debugger before running the program. This is useful when all your programs are compiled for debugging and you want to start your remote debugger at the very first line of the application.</li> </ul> <p>The default value is 0.</p> <p><b>Note:</b> This runtime property is used also by the compiler to let you debug preprocessor programs, if these programs are written in COBOL, through the remote debugger.</p>
jopaz.tracelevel=	<p>This property allows the user to define the events to be traced.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>■ 3 – Includes config settings and program starts and ends.</li> <li>■ 7 – Includes config, program starts/ends, and paragraph starts/ends.</li> <li>■ 11 – Includes config, program starts/ends, and file input/output.</li> <li>■ 15 – Includes config, program and paragraph starts/ends, and file i/o.</li> <li>■ 43 – Includes config, program starts/ends, and file input/output plus the content of read records.</li> <li>■ 63 – Traces everything of the above.</li> <li>■ 256 – Traces SQL activity.</li> </ul> <p>The default value is 0.</p>
jopaz.trace_location=	Specifies where to output the tracing information.

Option	Description
	<p><b>Note:</b> Set this property only if you set the trace level property <code>jopaz.tracelevel=</code>.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>■ 1 – Activates the writing of the tracing information into the job class STDOUT when the runtime job is executed.</li> <li>■ 2 – Activates the writing of the tracing information into a UNIX file under the path specified in the runtime property <code>jopaz.logfile=</code>.</li> <li>■ 3 – Combines option one and two.</li> </ul> <p>The default value is 1.</p>

## Configuring the Java Virtual Machine (jvm\_default\_conf.sh)

The JVM configuration file `jvm_default_conf.sh` contains the most important path settings for the Java Virtual Machine (JVM). Use this script file to correctly configure paths such as `LIBPATH`, `CLASSPATH`, and `LD_LIBRARY_PATH` and to ensure the inclusion of all necessary resources.

In addition, the correct mainframe encoding for JZOS is set in this script and the location of the corresponding Jopaz property file is transferred to the environment variable `IBM_JAVA_OPTIONS`.

### Db2 JDBC Location

Jopaz can execute Db2 SQL statements using JDBC (Java Database Connectivity). The Jopaz installation uses the following IBM standard paths for the archives and resources required by JDBC:

```
DB2=/usr/lpp/db2c10/jdbc/classes/db2jcc_javax.jar
DB2="${DB2}"/usr/lpp/db2c10/jdbc/classes/db2jcc_license_cisuz.jar
DB2="${DB2}"/usr/lpp/db2c10/base/lib/clp.jar
DB2="${DB2}"/usr/lpp/db2c10/jdbc/classes/db2jcc4.jar
DB2="${DB2}"/usr/lpp/db2c10/jdbc/classes/sqlj4.zip
```

If you have a different installation directory for Db2 and JDBC, you have to adjust these paths in the `jvm_default_conf.sh` file. Otherwise, a correct connection to the Db2 database from the Jopaz compiled application cannot be guaranteed.

The definitions of the JDBC class paths are set in the `get_default_db2_conf` function of `jvm_default_conf.sh`.

# 3 Jopaz for Adabas

---

■ About Jopaz for Adabas .....	52
■ Installing Jopaz for Adabas .....	52
■ Installing Adabas Service for Java .....	53
■ Jopaz for Adabas Runtime Configuration .....	53
■ Running the Installation Verification Program PGMADAOP.class .....	55

## About Jopaz for Adabas

---

This chapter describes the installation and runtime configuration of Jopaz for Adabas.

Jopaz for Adabas is an extension to Jopaz providing support for Cobol batch programs which use Adabas direct calls to access Adabas databases on z/OS. A requirement for this support is installing the Adabas Service for Java component which is included in the supplied Jopaz for Adabas package.

## Installing Jopaz for Adabas

---

This section describes the installation steps of Jopaz for Adabas.

1. Create an empty z/OS dataset with following attributes:

```
DSN=SAG.JPZADvrs.XMIT
Dataset Organisation=PS
SPACE=(CYL,(5,5,5))
DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
```

2. Using an FTP connection in binary mode, copy the XMIT file with the same DSN as the previously created dataset and paste it on your mainframe machine.  
  
A prompt appears asking you if the existing dataset should be overwritten.
3. Confirm the prompt.
4. Copy the JCL unpack job UPJPZADA.jcl using an FTP connection in ASCII mode to an accessible location on z/OS.

5. In the UPJPZADA.jcl job change:

- the location of the XMIT file. Look for SAG.JPZADvrs.XMIT.
- the location of the target dataset. Look for SAG.JPZADvrs.CONT.
- the *HLQ* in the RENAMEU statement.
- the CLASS and MSGCLASS parameters of the job header to their respective valid values. This depends on your machine configuration.

6. Run the job as a JES batch job.

After unpacking, you should find the HLQ.JPZADvrs.TAR dataset under the specified HLQ.

If the unpacking was successful, you can now delete the XMIT file and the target container dataset. They are no longer necessary for completing the installation.

7. Copy and uncompress the .tar.Z archive in the UNIX System Services filesystem with the following lines of code.

- a. Create a directory in the UNIX System Services filesystem to use as the JPZ installation directory.

```
$ mkdir -p /opt/softwareag
```

**Note:**

These instructions use /opt/softwareag as the default installation directory, but you must change it according to your system configuration.

- b. Copy the supplied JPZADvrs.tar.Z package to the installation directory.

```
$ cp "'HLQ.JPZADvrs.TAR(JPZADTAR)'" /opt/softwareag/JPZADvrs.tar.Z
```

- c. Extract the contents of JPZADvrs.tar.Z into the installation directory.

```
$ cd /opt/softwareag
$ uncompress < JPZADvrs.tar.Z | tar -oxf -
```

You should now see the jpz-ada-installer directory containing the following subdirectories:

- jpz-ada-cobol-templates
- jpz-ada-jcl-templates
- jpz-ada-uss-templates
- AdabasClient

## Installing Adabas Service for Java

Adabas Service for Java is an essential component for the successful operation of Jopaz for Adabas.

For information on installing the Adabas Service for Java, refer to the README.TXT file in the AdabasClient directory created as a result of “installing Jopaz for Adabas” on page 52 and follow the installation steps in the official documentation found at <https://documentation.softwareag.com/>.

The Adabas Service for Java installation process includes the creation of load library AJZvrs.LOAD and the installation of the AJZJVMP front end procedure, both of which are referenced later in this chapter.

## Jopaz for Adabas Runtime Configuration

This section describes the runtime configuration of Jopaz for Adabas.

To compile the sample Adabas COBOL program with Jopaz, you can use the compile job and environment from the default Jopaz installation. You can find the example program PGMADAOP.cbl at /jpz-ada-installer/jpz-ada-cobol-templates.

In the following steps, the installation guide uses HLQ.SAG.JPZvrs.JOBS as the Jopaz job library. This is the PDSE created for the native Jopaz installation. For reference to the Jopaz directory in the USS, the installation guide uses /opt/softwareag/jpz.

1. Copy the sample source PGMADAOP.cbl into the source directory:

```
$ cd /opt/softwareag
$ cd jpz-ada-installer/jpz-ada-cobol-templates
$ cp PGMADAOP.cbl "'HLQ.SAG.JPZvrs.SRCE'"
```

2. Copy the JVM configuration library into the conf directory:

```
$ cd /opt/softwareag
$ cd jpz-ada-installer/jpz-ada-uss-templates
$ cp template_jvm_ada_conf.sh /opt/softwareag/jpz/jpz-compiler/conf/
```

3. Copy the execution environment, the database definition file, and the execution job into Jopaz dataset. This is the Jopaz job library which was created for the Jopaz installation and mentioned in the variable input file. The installation instructions use HLQ.SAG.JPZvrs.JOBS.

```
$ cd /opt/softwareag
$ cd jpz-ada-installer/jpz-ada-jcl-templates
$ cp * "'HLQ.SAG.JPZvrs.JOBS'"
```

4. Adjust the database ID and the SVC number.

Rename the member HLQ.SAG.JPZvrs.JOBS(DB11111) to fit the database id you plan to access. For example, use DB00145 for DBID 145. Then edit and change the values in the member according to the site Adabas installation.

5. Check the member HLQ.SAG.JPZvrs.JOBS(AJZENV) and adjust the parameters for the Adabas Services for Java if necessary. Initial execution is also possible with the standard parameters. For a detailed description of these options, please check the official Adabas Services for Java documentation.
6. Adjust the paths in JZOS configuration HLQ.SAG.JPZvrs.JOBS(ADACONF) if you use other than the default:

```
readonly jpzlocation=/opt/softwareag/jpz/
"$jpzlocation"jpz-compiler/conf/template_jvm_ada_conf.sh
```

7. Adjust the paths in JVM configuration template\_jvm\_ada\_conf.sh located at /opt/softwareag/jpz/jpz-compiler/conf if you use other than the default:

```
export JPZ_INSTALLATIONPATH=/opt/softwareag/jpz/
export JAVA_HOME=/usr/lpp/java/J8.0_64
SAG=/opt/softwareag/jpz-ada-installer
ADALNKR=AJZvrs.LOAD
```

8. Adjust the DSN paths in samples execute job HLQ.SAG.JPZvrs.JOBS(ADAEXEC) if you use other than the default:

```
/* Adabas Service for Java front end procedure
// SET JVMADA=AJZJVMP
/* JZOS configuration for Jopaz for Adabas runtime
// SET CONFIG=HLQ.SAG.JPZvrs.JOBS(ADACONF)
/* Jopaz license file
// SET JPZLIC=HLQ.LICENSES(JPZBT11)
/* Adabas DB attributes file
```

```
// SET DDCARD=HLQ.SAG.JPZvrs.JOBS(DB11111)
//* Adabas Service for Java parameter file
// SET AJZENV=HLQ.SAG.JPZvrs.JOBS(AJZENV)
//* Adabas Service for Java load library
// SET AJZLIB=AJZvrs.LOAD
//* Adabas load library
// SET ADALIB=ADAvrs.LOAD
```

and

```
//STEPLIB DD DSN=HLQ.MLC137.MVSLOAD,DISP=SHR
// DD DSN=HLQ.JPZvrs.MVSLOAD,DISP=SHR
```

## Running the Installation Verification Program PGMADAOP.class

Now that you have everything installed and configured, you can submit the test job ADAEXEC. With this job the Java .class of the PGMADAOP test program compiled by Jopaz is executed. The PGMADAOP program contains two Adabas commands: OPEN and CLOSE.

If OPEN is successful, the program reports:

```
===== Adabas AJZ Interface =====
Installation Verification Program PGMADAOP
=====
Adabas OPEN(OP) Success. RSP: 0000
Adabas User -Queue Userid ....: COBBTCHU
-----
Adabas CLOSE(CL) Success. RSP: 0000
Total Commands Issued .....: 0000002
=====
```

If it fails, it reports:

```
===== Adabas AJZ Interface =====
Installation Verification Program PGMADAOP
=====
Adabas OPEN(OP) error. RSP: 0148
SUBC: F
=====
```





# 4 Jopaz for Db2

---

■ Installation Parameters for Db2 .....	58
■ Configuring Jopaz for Db2 .....	59
■ Setting up a Db2 Password .....	61
■ Preparing Programs for Static Execution .....	61
■ Executing Applications with Db2 Access .....	67

## Installation Parameters for Db2

This section describes all installation parameters that are available for Db2 in `jpz_variable_input_file.txt`.

**Note:**

You must define these parameters before installing Jopaz if you want Jopaz to be able to work with a Db2 database.

Parameter Name	Mandatory?	Description
JPZ_DB2	Yes	Specifies whether you want to use Jopaz in combination with a DB2 database.  Possible options are YES or NO.
JPZ_DB2_HOME	Yes	Specifies the DB2 installation location.  The IBM default is /usr/lpp/db2d10.
JPZ_DB2_NAME	Yes	Specifies the name of your DB2 database.
JPZ_DB2_SSID	Yes	Specifies the SSID of your DB2 database.  You can find the SSD in the DB2I Primary Option Menu, which is called up from the ISPF.
JPZ_DB2_URL	Yes	Specifies the URL of your DB2 database.
JPZ_DB2_USER	Yes	Specifies the username of your DB2 database.
JPZ_DB2_SERVER_NAME	Yes	Specifies the server name of your DB2 database.
JPZ_DB2_PORT	Yes	Specifies the port of your DB2 database.
JPZ_DBRM_DSN	Yes	Specifies the dataset (PDSE) name for the SQLJ DBRM modules. For example: HLQ.SAG.JPZVRS.DBRM
JPZ_DB2_UTILITY_DSN	Yes	Specifies the DSN load library containing the IBM utility DSNTEP2.  The IBM default is DSN1210.RUNLIB.LOAD.
JPZ_DB2_CTX_NAME	No	Specifies a trusted context name for JPZ_DB2_USER and generates the JCL job TRUSCONT, which creates a new trusted connection.  The name must be maximum 8 characters.  This parameter is optional and can be left blank if you do not need to use Trusted Context for Db2 authorization or if Trusted Context is already available for JPZ_DB2_USER.

Parameter Name	Mandatory?	Description
JPZ_DB2_CTX_PLAN	No	<p>Specifies a trusted context plan name to bind the utility DSNTEP2 and generates the JCL job TRUSCONT, which creates a new trusted connection.</p> <p>The name must be maximum 8 characters.</p> <p>This parameter is optional and can be left blank if you do not need to use Trusted Context for Db2 authorization or if Trusted Context is already available for JPZ_DB2_USER.</p>

## Configuring Jopaz for Db2

The following chapter lists components with which you can configure Jopaz for Db2 according to your specifications. If you entered all Db2 values when installing Jopaz, an initial configuration setup already exists after the installation.

## Setting up the Jopaz for Db2 Configuration File

To configure Jopaz for Db2, you need to specify the following parameters in the `jpz.properties` file present in `/jpz/jpz-compiler/conf` directory. These parameters control the properties of the client related configurations of Jopaz for Db2.

Parameter Name	Mandatory	Description
<code>java.home</code>	Yes	Specifies the Java JDK installation directory.
<code>db2.home</code>	Yes	Specifies the Db2 installation directory, which contains licenses, JDBC, and SQLJ drivers.
<code>jpz.maxClients</code>	Yes	Specifies the maximum number of clients that can connect simultaneously to server executing Db2 requests.
<code>jpz.initDb2Connection</code>	Yes	Specifies the number of initial connections to Db2.
<code>jpz.bufferLentgh</code>	Yes	<p>Specifies the length of the client buffer in MB.</p> <p>Possible values are integers from 1 to 5000.</p>
<code>jpz.homePath</code>	Yes	<p>Specifies the path or paths of static profiles.</p> <p>During the static preparation, serialized profiles are generated in the directory mentioned in this parameter. The profiles in the static path are used during the runtime to execute the SQL statements statically.</p>

## Setting up the Jopaz USS Environment Configuration File

You can find the configuration script `jvm_conf.sh` in `/jpp/jpz-compiler/conf`. This script is called by the configuration member `EXECCONF` and sets the necessary environment variables required for static preparation. Jopaz builds the environment variables like `CLASS_PATH` and `LIB_PATH` based on parameters you supply.

Parameter Name	Mandatory	Description
DB2_HOME	Yes	Updates the Db2 installation directory, which contains licenses,JDBC, and SQLJ drivers. For example: <code>/usr/lpp/db2vrs</code> .
JAVA_HOME	Yes	Updates the Java JDK installation directory. For example: <code>/usr/lpp/java/Jvrs</code> .
JPZ_HOME	Yes	Specifies the Jopaz installation directory path where your Jopaz files are located.

## Setting up the Db2 Configuration File

In order for Jopaz to be able to connect to Db2 and to perform SQL operations, you must set a few properties in the `db2.properties` file. Other than the parameters mentioned in this section, you can also specify JDBC and SQLJ properties in the `db2.properties` file.

Parameter Name	Mandatory	Description
user	Yes	Sets up the userid to connect to Db2. This user must require access on Db2 to perform SQL operations.
password	No	Sets up the password for the Db2 user.  This property is not mandatory, because the recommended method to store your password is through the Jopaz Db2 password encryption capability.
databaseName	Yes	Sets up the Db2 database location name to connect to the specific Db2 system. For example: <code>DAEFDB2D</code> .
serverName	Yes	Specifies the fully qualified domain address of the z/OS system.
portNumber	Yes	Specifies the TCP/IP port number, which identifies the specific Db2 subsystem.

---

## Setting up a Db2 Password

---

### Using a Db2 Encrypted Password

Jopaz for Db2 provides a password encryption mechanism to safely encrypt your password using encryption keys. It is recommended that you use this form of password authentication.

To encrypt the password, you need to generate an encryption key. You can generate the key and encrypt the password using the script `/j pz/j pz-scripts/j pz-db2-pass.sh`. When you execute the script, a prompt asks you to enter a Db2 connection password.

Use the option `-g` when you execute the script for the first time to generate an encryption keyfile.

```
j pz-db2-pass.sh -g
```

You can also use the option `-g` to change an existing encryption key. If you want to change the password without changing the keyfile, execute the script without `-g`.

If your Db2 connection password is encrypted, the password parameter in `db2.properties` is ignored.

**Note:**

You must use this password independently of the use of a trusted context. When setting up Jopaz for Db2, you need the password to customize and bind the serialized profile.

### Using a Db2 Non-Encrypted Password

As an alternative to the encrypted password, you can also specify the password as an additional property in the `db2.properties` file.

To specify the password, open the `db2.properties` file in `/j pz/j pz-compiler/conf` and add your Db2 password under the property `password=`.

---

## Preparing Programs for Static Execution

---

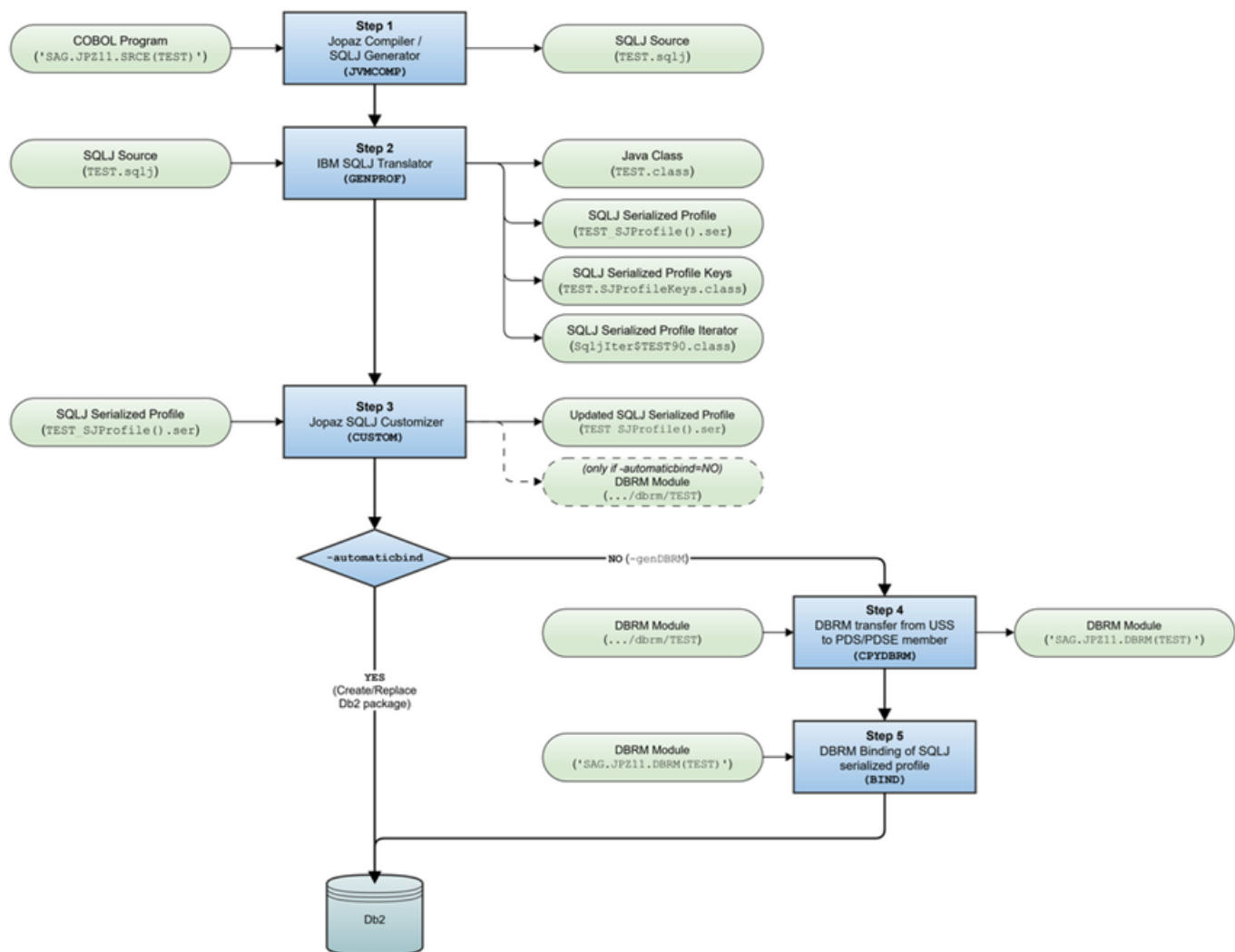
In order to execute COBOL programs that contain Db2 statements statically with Jopaz, you must perform the static generation steps in the following sections for each program you want to execute. This section provides an overview of the preparation process for static execution.

Jopaz uses Java to connect to Db2 and execute SQL statements. The standard for embedding SQL statements in Java programs is called SQLJ and the Java programs containing embedded SQL are called SQLJ programs. Jopaz provides the SQLJ generator JVMCOMP that translates temporary Java programs generated by the Jopaz compiler into SQLJ. To do that, you use the compiler option `-sqlj`.

You must then use an SQLJ translator provided by IBM. The translator receives your SQLJ program as input and produces several files including SQLJ serialized profiles. These profiles are files containing the description of the SQL statements and cursors from a particular program. Jopaz uses such profiles at runtime.

These SQLJ serialized profiles must be then customized and bound to Db2. Jopaz provides its own customizer program, which calls the IBM SQLJ customizer. The Jopaz customizer uses the connection parameters specified in the Jopaz configuration file `db2.properties` to execute the customization process in Db2. The recommended standard procedure provided in the job example JPZCOMPS is set to skip the binding process and instead create a DBRM file that is bound to Db2.

The DBRM file adapted from the serialised SQLJ profile and customized by the Jopaz SQLJ customizer must be bound to the Db2 database using the Db2 command `BIND PACKAGE`. This is particularly useful if you do not have the required permissions to execute the bind process in Db2. In this case, the bind can be performed with the permissions of the executor of the `BIND PACKAGE` command, or the user who submits the batch job containing the command. To execute the `BIND PACKAGE` step, the corresponding DBRM module generated in the Unix SystemService (USS) environment must first be copied into a PDS/PDSE member.



## Step 1 – Using the Jopaz SQLJ Generator JVMCOMP

Execute the Jopaz compiler Java class `com.jopaz.compiler.Pcc` with the IBM JZOS Batch Launcher and use the SQLJ-specific compiler properties file `COMOPTS` included in the Jopaz installation package.

### Compile Properties

```
# COMPILER SETTINGS
# There must be TWO SPACES before a line break "\""
jopaz.compiler.options=-b -cod1 -crv -la \
-od=/opt/softwareag/jpz/jpz-compiler/output -sqlj -csdb2 -csqq2
```

### JCL Example

```
//JVMCOMP EXEC PROC=&JVM64,REGSIZE=512M,
// JAVACLS='com.jopaz.compiler.Pcc',
// ARGS=&CLASS.
//JPZSRC DD DISP=SHR,DSN=&SRCLIB.(&CLASS.)
//JPZOPT DD DISP=SHR,DSN=&OPTLIB.(COMOPTS)
//SYSLIB DD DISP=SHR,DSN=&CPYLIB
//STDENV DD DISP=SHR,DSN=&CONFIG
//STEPLIB DD DISP=SHR,DSN=&MLCLIB
// DD DISP=SHR,DSN=&ASMLIB
//LICJPZBT DD DISP=SHR,DSN=&JPZLIC
```

## Step 2 – Using the IBM SQLJ Translator GENPROF

Execute the `sqlj.tools.Sqlj` Java class through the IBM JZOS Batch Launcher using the SQL program translated in the previous step as input. The Java class `sqlj.tools.Sqlj` is provided with the IBM JDBC/SQLJ driver for Db2 for z/OS. The IBM SQLJ translator has a similar function as a Db2 precompiler for Assembler or COBOL. The translator allows the embedded SQL statements in a program to be executed statically with Db2. The main difference is that the SQLJ translator generates both a serialized profile and a modified version of the input program as output. Another important difference is that the IBM SQLJ translator does not generate a DBRM file. For details about the IBM SQLJ translator, please refer to appropriate IBM documentation.

### JCL Example

```
// SET CLASS='HELLO'
// SET STATPATH='/opt/softwareag/jpz/jpz-compiler/output'
//*****
//GENPROF EXEC PROC=&JVM64,REGSIZE=1024M,COND=(0,NE,JVMCOMP.JAVAJVM),
// JAVACLS='sqlj.tools.Sqlj'
//MAINARGS DD *,SYMBOLS=JCLONLY
// compile=true -d=&STATPATH
// &STATPATH./&CLASS..sqlj
//STDENV DD DISP=SHR,DSN=&CONFIG
//STEPLIB DD DISP=SHR,DSN=&MLCLIB
// DD DISP=SHR,DSN=&ASMLIB
//LICJPZBT DD DISP=SHR,DSN=&JPZLIC
```

## Step 3 – Running the Jopaz SQLJ Customizer CUSTOM

Run the Java class `com.softwareag.jopaz.sqlj.Main` through the IBM JZOS Batch Launcher to customize the serialized profile generated in the previous step and, optionally, execute the Db2 bind. Jopaz provides the Java class `com.softwareag.jopaz.sqlj.Main` with the following options:

Option	Argument	Mandatory	Description
<code>-url</code>	Db2 connection URL in the format <code>jdbc:db2://server:port/database</code>	No	Specifies the URL connection to a Db2 server instance.  If you do not specify this option, the connection parameters available in the configuration file <code>db2.properties</code> are used instead. If you specify this option, you must also specify the options <code>-user</code> and <code>-password</code> .
<code>-user</code>	Db2 connection user ID	No	Specifies the connection user ID to authenticate a Db2 server instance.  If you do not specify this option, the user specified in the configuration file <code>db2.properties</code> is used instead. If you specify this option, you must also specify the options <code>-url</code> and <code>-password</code> .
<code>-password</code>	Db2 connection password	No	Specifies the connection user password to authenticate a Db2 server instance.  If you do not specify this option, the encrypted password or the password specified in the configuration file <code>db2.properties</code> is used instead. If you specify this option, you must also specify the options <code>-url</code> and <code>-user</code> .
<code>-automaticbind</code>	YES or NO	No	Specifies whether the bind is executed after the customization or not.  If you set this option to NO, you must use the options <code>-genDBRM</code> and <code>-DBRMDir</code> to execute the bind using the generated DBRM file.  The default value is YES.
<code>-genDBRM</code>	None	No	Specifies whether the class generates a DBRM file after execution.  When you specify this option, you must also specify <code>-DBRMDir</code> .



Option	Argument	Mandatory	Description
-DBRMDir	DBRM files directory	No	<p>Specifies the Unix System Services directory where the DBRM file generated from the serialized profile is stored. This is applicable only when you have the option -genDBRM specified.</p> <p>The generated DBRM file name is the value specified in the parameter -profile.</p> <p>You must specify this option with the option -genDBRM.</p>
-bindoptions	Db2 bind options	Yes	<p>Specifies the bind options used to bind the serialized profile to Db2.</p> <p>You must always specify the isolation level (ISOLATION CS/RS/RR/UR/NC).</p> <p>If you specified -automaticbind YES, you can also specify additional bind options.</p>
-profile	Serialized profile name	Yes	<p>Specifies a name prefix for the generated serialized profile.</p> <p>This file is created in the directory specified under the property jopaz.staticPath in the jpz.properties file.</p> <p>The generated profile file name has the format of the serialized profile name_SJProfile0.ser.</p> <p>It is recommended to use the same name as the .class file.</p>
-collection	Db2 collection name	No	<p>Specifies the collection where the packages from the bind process are added.</p>
-staticpositioned	YES or NO	No	<p>Specifies whether positioned update statements execute statically or dynamically.</p> <p>It is recommended to set this option to YES.</p>
-onlinecheck	YES or NO	No	<p>Specifies whether online checking of data types is performed against the Db2 instance from the file db2.properties or from the -url option.</p> <p>It is recommended to set this option to YES.</p> <p>When -onlinecheck YES, the user</p>

Option	Argument	Mandatory	Description
			must have authorization to execute the statements, which are customized.
-pkgversion	AUTO or version-id	No	Specifies which package version is used when packages are bound to the server for the customization of the serialized profile.  If this parameter is not specified, no version is used.
-qualifier	Qualifier name	No	If -onlinecheck YES is set, specifies the qualifier used for unqualified objects in the SQLJ program during online checking.

## JCL Example

```
// SET CLASS='HELLO'
// SET DBRMPATH='/opt/softwareag/jpz/jpz-compiler/dbrm'
// SET COLL=TSTJPZ
// SET ISO=CS
// *-----
//CUSTOM EXEC PROC=&JVM64,REGSIZE=1024M,
//      JAVACLS='com.softwareag.jopaz.sqlj.Main',
//      COND=(0,NE,GENPROF.JAVAJVM)
//MAINARGS DD *,SYMBOLS=JCLONLY,LRECL=150
-profile &CLASS -bindoptions ISOLATION &ISO
-onlinecheck YES -staticpositioned YES
-collection &COLL -automaticbind NO
-genDBRM -DBRMDir &DBRMPATH
//STDENV DD DISP=SHR,DSN=&CONFIG
//STDERR DD SYSOUT=*
//STDOUT DD SYSOUT=*
```

If -automaticbind NO and -genDBRM and -DBRMDir options are specified, the class doesn't execute the binding of the serialized profile to Db2. The class creates a DBRM file in the directory specified by -DBRMDir option.

You can copy the DBRM file to a dataset and bind it to Db2 by executing the Db2 `BIND PACKAGE` command. For more information about the Db2 `BIND PACKAGE` command, see the IBM Db2 for z/OS SQL Reference.

If -onlinecheck YES, the user specified in the db2.properties file or the user specified in the -user parameter must have access to execute the statements when customizing the the SQLJ program.

## Step 4 – Transferring DBRM from USS to a PDS/PDSE Member

Use the IBM utility `IKJEFT01` to execute the `TSO OCOPY` copy command in the batch job.

Enter the created and customized DBRM module from Step 3 as input in this step. This only needs to be transferred to a PDS/PDSE dataset in order to be able to execute `BIND PACKAGE`.

## JCL Example

```
// SET CLASS='HELLO'
// SET DBRMPATH='/opt/softwareag/jpz/jpz-compiler/dbrm'
// *-----
//CPYDBRM EXEC PGM=IKJEFT01,
//          COND=((0,NE,CUSTOM.JAVAJVM),(0,NE,GENPROF.JAVAJVM))
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
//          OCOPY INDD(INDD) OUTDD(OUTDD) BINARY
// *
//INDD      DD PATH='&DBRMPATH/&CLASS'
//OUTDD     DD DISP=SHR,DSN=&DBRMLIB(&CLASS)
```

## Step 5 – DBRM Binding of SQLJ Serialized Profile with BIND

Bind the generated DBRM module to Db2 using the Db2 `BIND PACKAGE` command. This step must be completed when both `-automaticbindis` set to `NO` and `-genDBRM` is specified in Step3. For more information about the Db2 `BIND PACKAGE` command, refer to the IBM Db2 documentation.

## JCL Example

```
// SET CLASS='HELLO'
// SET COLL=TSTJPZ
// SET ISO=CS
// SET DB2SSID=<JPZ_DB2_SSID>
// SET DB2NAME=<JPZ_DB2_NAME>
// SET DBRMPATH='/opt/softwareag/jpz/jpz-compiler/dbrm'
// *-----
//BIND      EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=4096K,
//          COND=((0,NE,CUSTOM.JAVAJVM),(0,NE,GENPROF.JAVAJVM),
//          (0,NE,CPYDBRM))
//STEPLIB   DD DISP=SHR,DSN=SYS1.DSN1310.SDSNLOAD
//DBRMLIB   DD DISP=SHR,DSN=&DBRMLIB
//SYSTSPRT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//SYSTSIN   DD *,SYMBOLS=JCLONLY
//          DSN SYSTEM(&DB2SSID)
//          BIND PACKAGE(&DB2NAME..&COLL) -
//          MEM(&CLASS) -
//          ENCODING(273) -
//          ISOLATION(&ISO) RELEASE(COMMIT) EXPLAIN(NO) -
//          ACTION(REPLACE) APPLCOMPAT(V12R1M510)
//          END
// *
```

## Executing Applications with Db2 Access

To execute Jopaz-translated COBOL programs containing Db2 instructions, you must decide how to authenticate your applications against the Db2 database.

The default option is to create a trusted context and to establish a trusted connection. With a trusted context, connections from certain users or applications are allowed to bypass conventional user

authentication like password-based authentication because of a trusted relationship between the client and the database.

The second option is to verify yourself against the database using conventional user authentication. To do this, the correct password must be stored in the Jopaz configuration and the user must have authorization on the Db2 database to be able to carry out the operations.

## Authentication Using a Trusted Context

To use a trusted context when authenticating against the database, you must set the Jopaz Runtime property `jopaz.jdbc.datasource` as follows:

```
jopaz.jdbc.datasource=JPZDataSourceTrustedContext
```

This is not part of the standard Jopaz setup, so you must set this manually. The runtime properties are set in the RUNOPT member of the Jopaz JOBS library.

### Creating a Trusted Connection

If you do not have a trusted connection established or you want to create a new one, you must use the TRUSCONT JCL job. To use the job successfully, you need to have administrative rights to the database, or DBADMAUTH permissions with option GRANT set.

Before you submit the job, open it and make sure that all parameters are set correctly. They are already set according to the `jpz_variable_input_file.txt` file during Jopaz installation. If there are differences, you must adjust the job.

Submit the job to create the trusted context. If the job has been executed without an error message, a new trusted context is created.

#### Note:

If you execute TRUSCONT for the first time without a Trusted Context with this name previously existing, the job ends with an RC=0008, because the DROP step of the possibly existing Trusted Context cannot be executed. This behaviour is acceptable and you do not have to take any actions.

## Authentication Using a Username and Password

To use a username and password to authenticate against the database, you must set the Jopaz Runtime property `jopaz.jdbc.datasource` as follows:

```
jopaz.jdbc.datasource=JPZDataSource
```

This is the Jopaz default setting. If you want to use this option, you do not need to adjust anything else.

The runtime properties are set in the RUNOPT member of the Jopaz JOBS library.

The DataSource `JPZDataSource` option automatically uses the password you set during password encryption. For more information, see [“Setting up a Db2 Password” on page 61](#). Both encrypted and a non-encrypted passwords are supported.

# 5 Compiling with Jopaz

---

■	Compiling on Windows .....	70
■	Compiling on z/OS .....	70
■	Remote Debugging with Jopaz .....	79

## Compiling on Windows

---

To compile on Windows using Jopaz you must:

1. Create a new project by selecting **File > New > Jopaz Project**.
2. Enter a project name and click **Finish**.
3. Switch to the **File** tab.
4. Copy the Cobol source code into the subfolder **source**.
5. Select one of the following methods to start compiling:

- Right click on the program and select **Compile**.
- From the navigation bar, select the **C** icon.

The new Java programs are saved in folder **output**.

## Compiling on z/OS

---

You can find two sample JCL jobs in the `jpz-jcl-templates` folder from the delivery package to enable you to compile on z/OS. `JPZCOMP` allows the compilation from Cobol to Java using the Jopaz compiler. `JPZEXEC` then executes the Java class (.class) compiled from the Jopaz compiler directly with JZOS. Both jobs are for Java in 64-bit addressing mode.

Each job uses a configuration file that you can find in the `jpz-jcl-templates` folder. If the installer script was used at the beginning, these two files were also automatically copied to the desired dataset and adapted to the installation. `COMPCONF` is the Jopaz configuration file for compilation. `EXECCONF` is the Jopaz configuration file for execution.

Accordingly, the references look like:

- `JPZCOMP`
  - `&JVM64 = JVMPRC86`
  - `&CONFIG = COMPCONF`
- `JPZEXEC`
  - `&JVM64 = JVMPRC86`
  - `&CONFIG = EXECCONF`

Both configuration files are necessary for their respective jobs. They use a configuration library, which is stored in the UNIX system services under the directory `/jpz-compiler/conf/jvm_default_conf.sh`. This file contains a number of standard configurations that can be included in the configuration files by means of method calls.

The examples of COMPCONF and EXECCONF shown here demonstrate how the standard configurations can also be overwritten.

**Note:**

Alternatively, the configuration and execution scripts can be stored in a USS directory like /j pz/j pz-scripts and referenced via path specification. To do this, specify directly in the JCL job the absolute path to the file in quotation marks. For example, in JPZCOMP:

```
SET CONFIG = '/.../j pz-scripts/COMPCONF.sh'
```

For testing the setup, you can use the HELLO.cbl program included in the delivery package.

**Note:**

Regarding the CVE-2021-44228 Apache Log4j vulnerability, we have added the JVM option "-Dlog4j2.formatMsgNoLookups=true" to all configuration files to mitigate it.

## Compiling and Running with Jopaz using JZOS

All paths in the following templates must match the paths defined in the Jopaz configuration properties file.

### JPZCOMP - Jopaz Compile Job

```
//JPZCOMP JOB, CLASS=G,MSGCLASS=X,NOTIFY=&SYSUID
//*- - - - -
- - - - -
/* Batch Java VM Skeleton Job for JZOS using z/OS included scripts
//*- - - - -
- - - - -
//EXPORT SYMLIST=(*)
//*- - - - -
- - - - -
/*Section to set parameters for the job
//*- - - - -
- - - - -
// SET CLASS='HELLO'
// SET JVM64=JVM64
// SET CONFIG=JPZ_JOBS_DSN(COMPCONF)
// SET OPTLIB=JPZ_JOBS_DSN
// SET JPZLIC=JPZ_LICENSE_FILE
// SET SRCLIB=JPZ_SRCE_DSN
// SET CPYLIB=JPZ_COPY_DSN
//*- - - - -
- - - - -
/* Configure Java and Compile .cbl/.cob to .class
//*- - - - -
- - - - -
//JVMCOMP EXEC PROC=&JVM64,REGSIZE=512M,
//JAVACLS='com.jopaz.compiler.Pcc',
//ARGS=&CLASS.
//JPZSRC DD DISP=SHR,DSN=&SRCLIB(&CLASS.)
//JPZOPT DD DISP=SHR,DSN=&OPTLIB(COMOPT)
//SYSLIB DD DISP=SHR,DSN=&CPYLIB
//STDENV DD DISP=SHR,DSN=&CONFIG
//STEPLIB DD DISP=SHR,DSN=JPZ_LICENSE_CHECKER
```

```
//          DD DISP=SHR,DSN=JPZ_ASM_LOAD
//LICJPZBT DD DISP=SHR,DSN=&JPZLIC
//* -----
- - - - -
/* (Recommended) Set .class permissions automatically
/* -----
- - - - -
//SETPERM EXEC PGM=BPXBATCH
//STDPARM DD *,SYMBOLS=JCLONLY
SH
. JPZ_INSTALLATIONPATH/jpz-compiler/conf/jvm_default_conf.sh;
set_permissions;
//STDERR DD SYSOUT=*
```

## COMPCONF - Compile Configuration

```
readonly jpzlocation=JPZ_INSTALLATIONPATH
. "$jpzlocation"/jpz-compiler/conf/jvm_default_conf.sh
#####
#####
# you can comment out a line with '#' if you want to provide your #
# own config or leave a line uncommented to use the default config #
# the default_conf.sh file above in USS can be seen as a #
# reference for your own customizations #
#####
#####
get_default_lib_path
get_default_compiler_conf
get_default_db2_conf
get_default_JVM_compile_conf
/*
```

## COMOPT - Compiler Properties

```
# COMPILER SETTINGS
# There must be TWO SPACES before a line break "\"
jopaz.compiler.options=-b -codl -crv -la \
-od=<baseDir>/output
# ALTERNATIVE ADABAS CALL NAME IN SOURCE
# jopaz.compiler.adaname=literal
```

## JPZEXEC - Jopaz Execute Job

```
//JPZEXEC JOB ,CLASS=G,MSGCLASS=X,NOTIFY=&SYSUID
/*-----
/* Batch Java VM Skeleton Job for JZOS using z/OS included scripts
/*-----
// EXPORT SYMLIST=(*)
/*-----
/* Section to set parameters for job
/*-----
// SET !\colorbox{markYellow}{CLASS}!= 'HELLO'
// SET !\colorbox{markGreen}{JVM64}!=JPZ_JVMPROC
// SET !\colorbox{markBlue}{CONFIG}!=JPZ_JOBS_DSN(!\hyperref[JZOS:EXECCONF]{EXECCONF}!)
// SET !\colorbox{markGreen2}{OPTLIB}!=JPZ_JOBS_DSN
// SET !\colorbox{markRed}{JPZLIC}!=JPZ_LICENSE_FILE
/*-----
```



```

/* Configure Java and Compile .cbl/.cob to .class
/*-----
//JVMEXEC EXEC PROC=!\colorbox{markGreen}{\&JVM64}!,REGSIZE=512M,
// JAVACLS=!\colorbox{markYellow}{\&CLASS}!
//STDENV DD DISP=SHR,DSN=!\colorbox{markBlue}{\&CONFIG}!
//JPZOPT DD
DISP=SHR,DSN=!\colorbox{markGreen2}{\&OPTLIB}!(!\hyperref[JZOS:runtime_properties]{RUNOPT}!)
//STEPLIB DD DISP=SHR,DSN=JPZ_LICENSE_CHECKER
// DD DISP=SHR,DSN=JPZ_ASM_LOAD
//LICJPZBT DD DISP=SHR,DSN=!\colorbox{markRed}{\&JPZLIC}!,
//JPZPRINT DD SYSOUT=*
/*
/* Examples
/*
/* 1. ACCEPT parameters from STDIN
/*
/* //STDIN DD *
/* parameter1
/* parameter2
/* /*
/*
/* 2. Pass LINKAGE SECTION parameters using MAINARGS
/*
/* Parameters can be specified as literals or JCL variables.
/* Use single-quotes to delimit parameters containing spaces.
/*
/* // SET PARM1='parameter 1'
/* //MAINARGS DD *,SYMBOLS=JCLONLY
/* '&PARM1'
/* parameter2
/* /*
/*

```

## EXECCONF - Execute/Runtime Configuration

```

readonly jpzlocation=JPZ_INSTALLATIONPATH
. "$jpzlocation"/jpz-compiler/conf/jvm_default_conf.sh
#####
#####
# you can comment out a line with '#' if you want to provide your #
# own config or leave a line uncommented to use the default config #
# the default_conf.sh file above in USS can be seen as a #
# reference for your own customizations #
#####
#####
get_default_lib_path
get_default_db2_conf
get_default_runtime_dependencies
get_default_JVM_execute_conf
/*

```

## RUNOPT - Runtime Properties

```

# RUNTIME SETTINGS
# DB2
jopaz.jdbc.url=jdbc:db2://db2-url
jopaz.jdbc.driver=com.ibm.db2.jcc.DB2Driver
jopaz.jdbc.options=user=db2-user,password=db2-pw

```

```
jopaz.jdbc.auto_connect=1
# TRACELEVEL 15 INCLUDES CONFIG, PROGRAM/PARAGRAPH STARTS/ENDS AND FILE I/O
#jopaz.tracelevel=15
# SETTING THAT LEADING ZEROES ARE SHOWN
jopaz.gui.show_zeroes=1
# OPTION TO ACTIVATE THE CHECK THE DIVISION BY ZERO
jopaz.checkdiv=1
# OPTION TO ACTIVATE THE ARRAY OUT OF BOUNDS CHECK
# (DEFAULT = 0 = NO BOUNDARY CHECK | 1 = BOUNDARY CHECK)
jopaz.array_check=1
# Set SAGSort as "SORT" routine
jopaz.sort=com.softwareag.jopaz.sort.SAGSort
```

## jvm\_default\_conf.sh - Configuration Library on USS

```
# This is a shell script which configures
# any environment variables for the Java JVM.
# Variables must be exported to be seen by the launcher.
. /etc/profile
#####
###          base variables          ###
#####
# Jopaz install location
export JPZ_INSTALLATIONPATH=JPZ_INSTALLATIONPATH
# Jopaz output directory for Java classes
export JPZ_OUTPUTDIRECTORY=JPZ_OUTPUTDIRECTORY
# Java install location
export JAVA_HOME=JPZ_JAVALOCATION64
# CLASSPATH variable for later use
export CLASSPATH="$CLASSPATH"
#####
###          check enviroment          ###
#####
if [ ! -f ${JPZ_INSTALLATIONPATH}jpx-jar/jopaz.jar ]
then
    echo "ERROR: No Jopaz compiler found at: ${JPZ_INSTALLATIONPATH}jpx-jar/jopaz.jar."
    echo "==> wrong JPZ_INSTALLATIONPATH in jvm_default_config.sh"
    exit 1
fi
#####
###          default config functions          ###
#####
get_default_lib_path () {
    export PATH=/bin:${JAVA_HOME}/bin
    LIBPATH=/lib:/usr/lib:${JAVA_HOME}/bin
    LIBPATH="$LIBPATH:${JAVA_HOME}/lib/s390x
    LIBPATH="$LIBPATH:${JAVA_HOME}/lib/s390x/j9vm
    LIBPATH="$LIBPATH:${JAVA_HOME}/bin/classic
    LIBPATH="$LIBPATH:${JPZ_INSTALLATIONPATH}jpx-lib
    export LIBPATH="$LIBPATH":
    export LD_LIBRARY_PATH="$LIBPATH"
}
get_default_compiler_conf () {
    APP1=${JPZ_INSTALLATIONPATH}jpx-jar/jopaz.jar
    APP1=${APP1}:${JPZ_INSTALLATIONPATH}jpx-jar/jpx-tools.jar
    APP1=${APP1}:${JPZ_INSTALLATIONPATH}jpx-jar/lic-utils.jar
    CLASSPATH=${CLASSPATH}:${APP1}
    CLASSPATH=${CLASSPATH}:${JAVA_HOME}/lib:${JAVA_HOME}/lib/ext
}
}
```

```

get_default_db2_conf () {
    # DB2 specification example  ### PLEASE REVIEW AND POSSIBLY ADAPT ###
    DB2=/usr/lpp/db2c10/jdbc/classes/db2jcc_javax.jar
    DB2="${DB2}:/usr/lpp/db2c10/jdbc/classes/db2jcc_license_cisuz.jar
    DB2="${DB2}:/usr/lpp/db2c10/base/lib/clp.jar
    DB2="${DB2}:/usr/lpp/db2c10/jdbc/classes/db2jcc4.jar
    DB2="${DB2}:/usr/lpp/db2c10/jdbc/classes/sqlj4.zip
    CLASSPATH="${CLASSPATH}":"${DB2}"
}

get_default_runtime_dependencies () {
    compiler="${JPZ_INSTALLATIONPATH}"jpz-jar/jopaz.jar
    compiler="$compiler":"${JPZ_INSTALLATIONPATH}"jpz-jar/jpz-tools.jar
    compiler="$compiler":"${JPZ_INSTALLATIONPATH}"jpz-jar/lic-utils.jar
    compiler="$compiler":"${JPZ_INSTALLATIONPATH}"jpz-jar/jpz-apis.jar
    output="${JPZ_OUTPUTDIRECTORY}"jpz-compiler/output/
    CLASSPATH="$CLASSPATH":"$output":"$compiler"
    CLASSPATH="$CLASSPATH":"${JAVA_HOME}/lib":"${JAVA_HOME}/lib/ext
    APP_HOME=""
    # Add Application required jars to end of CLASSPATH
    for i in "${APP_HOME}"/*.jar
    do
        CLASSPATH="$CLASSPATH":"$i"
    done
}

get_default_JVM_execute_conf () {
    # Set JZOS specific options
    # Use this variable to specify enc
    export JZOS_OUTPUT_ENCODING=Cp1047

    # Configure JVM options
    IJO=" -Djopaz.conf.mvs=1"
    # Uncomment the following to aid in debugging "Class Not Found" problems
    #IJ1=" -verbose:class"
    # Uncomment the following if you want to run with Ascii file encoding..
    #IJ2=" -Dfile.encoding=ISO8859-1"
    export IBM_JAVA_OPTIONS="$IJO"
}

get_default_JVM_compile_conf () {
    # Configure JVM options
    IJO=" -Djopaz.conf.mvs=1"
    # Uncomment the following to aid in debugging "Class Not Found" problems
    #IJ1=" -verbose:class"
    # Uncomment the following if you want to run with Ascii file encoding..
    #IJ2=" -Dfile.encoding=ISO8859-1"
    export IBM_JAVA_OPTIONS="$IJO"
}

#####
###          helper functions          ###
#####

set_permissions () {
    find "${JPZ_OUTPUTDIRECTORY}"jpz-compiler -user $(whoami) | xargs chmod 770
}

```

## Passing Parameters in the JCL Job

You can pass parameters to Jopaz-compiled COBOL applications in two ways: using the STDIN DD card or the MAINARGS DD card.

If you chose the first method, you define the parameters using the `STDIN DD` card, which the COBOL application reads as `ACCEPT` values.

If you choose to use the `MAINARGS DD` card method, you pass the parameters to the `LINKAGE SECTION` of the Jopaz-compiled COBOL program.

When you are using JCL symbolic parameters, you must follow these rules:

- At the beginning of the job, all symbolic parameters need to be exported. The command for this is `EXPORT SYMLIST=(*)`.
- Specify the assigned value in single quotation marks. For example: `SET PARM1='parameter 1'`.
- In the definition of the `STDIN DD` or `MAINARGS DD` card, you must enable the use of JCL symbolic parameters in the input of the DD card with `*`, `SYMBOLS=JCLONLY`.
- When using the `MAINARGS DD` method, you must specify the JCL symbolic parameter in single quotation marks: `'&PARM1.'`.
- If you are using the `MAINARGS DD` method and you want to pass several JCL symbolic parameters, you must enclose them all in single quotation marks: `'&PARM1.&PARM2.'`.

### Accepting Parameters from STDIN

```
// EXPORT SYMLIST=(*)
...
// SET PARM1='parameter 1'
...
//STDIN    DD *,SYMBOLS=JCLONLY
&PARM1.
parameter2
```

### Passing LINKAGE SECTION Parameters Using MAINARGS

```
// EXPORT SYMLIST=(*)
...
// SET PARM1='parameter 1'
...
//MAINARGS DD *,SYMBOLS=JCLONLY
'&PARM1.'
parameter2
```

### Example with STDIN and MAINARGS

#### COBOL Program

```
IDENTIFICATION DIVISION.
    PROGRAM-ID. PARM1.
DATA DIVISION.
LINKAGE SECTION.
    01 PARM-GRP.
        05 PARM-LENGTH    PIC S9(04) COMP.
        05 PARM-DATA1     PIC X(5).
        05 PARM-DATA2     PIC 9(4).
```

```

WORKING-STORAGE SECTION.
01 PARM-DATA3          PIC X(4).
PROCEDURE DIVISION USING PARM-GRP.
    DISPLAY "MAINARGS LENGTH: " PARM-LENGTH.
    DISPLAY "MAINARGS ARGS1:  " PARM-DATA1.
    DISPLAY "MAINARGS ARGS2:  " PARM-DATA2.
    ACCEPT PARM-DATA3 FROM SYSIN.
    DISPLAY "STDIN ARGS3:     " PARM-DATA3.
    STOP RUN.

```

## Jopaz Execution JCL Job

```

...
// EXPORT SYMLIST=(*)
...
// SET CLASS='PARM1'
...
// SET ARGS1='Jopaz'
// SET ARGS2='1234'
// SET ARGS3='Test'
...
//JVMEXEC EXEC PROC=&JVM64,REGSIZE=512M,LOGLVL='+T',
// JAVACLS=&CLASS
//STDENV   DD DISP=SHR,DSN=&CONFIG
//JPZOPT   DD DISP=SHR,DSN=&RUNOPT
//STEPLIB  DD DISP=SHR,DSN=&MLCLIB
//          DD DISP=SHR,DSN=&ASMLIB
//LICJPZBT DD DISP=SHR,DSN=&JPZLIC
//*
//STDIN     DD *,SYMBOLS=JCLONLY
&ARGS3.
//*
//MAINARGS DD *,SYMBOLS=JCLONLY
'&ARGS1.&ARGS2.'

```

## STDOUT of JCL Job

```

MAINARGS LENGTH: 0009
MAINARGS ARGS1:  Jopaz
MAINARGS ARGS2:  1234
STDIN  ARGS3:    Test

```

## SYSOUT of JCL Job with Java Trace

This information is only visible if you have LOGLVL='+T' set.

```

JVMJZBL2004N Log level has been set to: T
JVMJZBL2999T -> JzosVM()
...
JVMJZBL2999T -> invokeMain()
JVMJZBL2999T javaClassName: 'PARM1'
JVMJZBL2999T Arg 1='Jopaz1234'
JVMJZBL1023N Invoking PARM1.main()...
JVMJZBL1056I Arguments to main...
JVMJZBL1057I Jopaz1234
JVMJZBL2999T -> JniUtil.convert()
JVMJZBL2999T <- JniUtil.convert()

```

```
JVMJZBL2999T JvmExitHook entered with exitCode=0, javaMainReturnedOrThrewException=0  
JVMJZBL1043N The Java virtual machine completed with System.exit(0)
```

# Remote Debugging with Jopaz

---

■ Remote Debugging on Eclipse .....	83
-------------------------------------	----

## Preparation for Remote Debugging

1. It is recommended to split the properties file into three separate files:
  - COMOPT = compiler options and all necessary settings for correct debugging
  - RUNOPT = only the Jopaz options that are necessary for execution
2. Link the new files correctly in your configuration files for the compile & execute environment:
  - In the compiling environment, change the file path for export IBM\_JAVA\_OPTIONS= and the file path for export JZOS\_MAIN\_ARGS=.
  - In the executing environment, change only the file path for export JZOS\_MAIN\_ARGS=.
  - In both compiling and executing environments, add the JOPAZ= variable with the path to the directory where your license file is located.
  - Add the \$JOPAZ option to your CLASSPATH in both environment configurations.
3. Add the compiler option -d or -dx to your COMOPT in order to be able to debug this program after compilation.

- -d is used to include the debug information. An additional class file is generated to store the debug information. The resulting classes don't include the source code. Source files must be available to the debugger during the debug session.
- -dx enables extended debugger functions. This option implies -d. In addition to the standard debugging features, all the variables in the class are generated including the ones that are not used in the program. Additionally, the literal constants that are generated during the execution and not as static fields in the generated class, are created as well.

When a program is compiled with -dx, the debugger is able to query and set all items of the program Data Division including the items that are not used in the Procedure Division. The IDE allows the source code to be changed while debugging.

With -dx the debugger is able to skip statements through the jump command. The resulting classes don't include the source code.

Source files must be available to the debugger during the debug session.

4. Recompile the source again with the updated compile environment that includes the debug compiler options described before.
5. For remote debugging, a few parameters must also be added to the RUNOPT file:

```
jopaz.rundebbug=2
```

- =0 – The program starts in run mode and the Remote Debugger is not active.
- =1 – The program starts in run mode and the Runtime Framework listens for connections from the Remote Debugger.



- =2 – The program starts in debug mode and the Runtime Framework waits for connections from the Remote Debugger.

```
jopaz.debug.port=9998
```

- This property specifies the port used by the Remote Debugger.
- The default value is 9999.



# Remote Debugging on Eclipse

## Configuring the Remote Debugging Client

1. Open Eclipse.
2. Open the Jopaz perspective by selecting **Window > Perspective > Open > Perspective > Other > Jopaz**.

**Note:**

Check that you have installed a relatively new version of the Jopaz plugin. For more information, check [“Installing Jopaz” on page 21](#).

3. Open the debug perspective in Eclipse through **Window > Perspective > Open Perspective > Other > Debug**.
4. Configure the remote debug connection as follows:
  - a. Open the menu option **Run > Debug Configurations > Jopaz Remote Application**.
  - b. Give the configuration a logical name.
  - c. Enter the hostname of the Mainframe machine.
  - d. Enter the port that you have already defined in RUNOPT.
  - e. Save your configuration by clicking **Apply**.
5. You need to configure the correct encoding for the representation of the source to be debugged:
  - a. Navigate to **Window > Preferences > General > Workspace**.
  - b. Change the text file encoding to Other: **IBM-1047**.

## Remote Debugging

1. Open the execution job of your application that uses the RUNOPT file with the debug configurations.

**Note:**

It is recommended to use an interface other than Eclipse, such as Terminal, ZOWE, or Web Com-Plete.

2. Open the debug configuration settings through **Run > Debug Configurations > Jopaz Remote Application > *your\_saved\_configuration***

where *your\_saved\_configuration* is the name you have given to your configuration settings file.

3. Start the remote debugging process by clicking **Debug**.

The debugger waits for a few seconds for you to start your job.

4. Start your job immediately.

On the sysout of the JVM, you should get this message: Debugger listening on port 9999  
....

If you have not started the job while the debugger is waiting, an error pop-up appears.

5. The debugging mode starts and a new instance of your configuration is created.

You can see the instance in the **Debug** tab.

6. When the debugging instance is started, you can change the format of the source code in the debugging mode by selecting **COBOL Source > Format > *your preferred appearance***.
7. You can now perform the desired operations, such as setting breakpoints and stepping through the single instructions.

**Note:**

For the next remote debugging run, you no longer need to go through the debug configuration menu. You can simply start the debugger through the bug icon in the toolbar.

## Switching Between Debugging Interfaces

In Eclipse, you can switch between two remote debugging interfaces. The default interface is the internal debugger which uses the standard eclipse debug perspective and its functionalities. The second interface is the external Jopaz debugger that uses the graphical debugger interface from Jopaz. It opens in a separate window when the debugging process is started.

To start using the external debugger:

1. Go to **Window > Preferences > Jopaz > Run/Debug**.
2. Set the option **Use external Jopaz Debugger**.

When you start the remote debugging session, the external Jopaz remote debugger window pops-up.

## 6 Optimizing DFSORT

---

■ Runtime Settings .....	86
■ Faster Compiling .....	86

## Runtime Settings

---

To enhance the program execution of DFSORT, you can use one of the options DFSPARM or SORTCNTL. These options can be defined in the JCL job as follows:

```
//DFSPARM DD *  
  OPTION EQUALS  
  OPTION MAINSIZE=MAX  
  OPTION MOSIZE=MAX,HIPRMAX=OPTIMAL,DSPSIZE=MAX
```

or

```
//SORTCNTL DD *  
  OPTION EQUALS  
  OPTION MAINSIZE=MAX  
  OPTION MOSIZE=MAX,HIPRMAX=OPTIMAL,DSPSIZE=MAX
```

## Faster Compiling

---

By converting to non-procedural COBOL code, you improve the EXCP and the job elapsed time. The programmatic implementation of the non-procedural COBOL code can be seen in the following code snippet:

```
ID DIVISION.  
PROGRAM-ID. PGMSORT.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT SORTIN  ASSIGN TO SORTIN.  
    SELECT SORTOUT ASSIGN TO SORTOUT.  
    SELECT SORT-FILE ASSIGN TO SORTFILE.  
DATA DIVISION.  
FILE SECTION.  
FD SORTIN RECORD CONTAINS 160 CHARACTERS  
  LABEL RECORD STANDARD BLOCK 16000  
  DATA RECORDS ARE SORTIN-RECORD.  
01 SORTIN-RECORD.  
  05 FILLER          PIC X(160).  
FD SORTOUT RECORD CONTAINS 160 CHARACTERS  
  LABEL RECORD STANDARD BLOCK 16000  
  DATA RECORDS ARE SORTOUT-RECORD.  
01 SORTOUT-RECORD.  
  05 FILLER          PIC X(160).  
SD SORT-FILE RECORD CONTAINS 160 CHARACTERS  
  DATA RECORD SORT-RECORD.  
01 SORT-RECORD.  
  05 SORT-KEY        PIC X(8).  
  05 FILLER          PIC X(152).  
WORKING-STORAGE SECTION.  
PROCEDURE DIVISION.  
MASTER SECTION.  
    SORT SORT-FILE  
    ON DESCENDING KEY SORT-KEY  
    USING SORTIN  
    GIVING SORTOUT.  
    IF SORT-RETURN > 0
```

```
DISPLAY "SORT FAILED WITH RC:" SORT-RETURN.  
STOP RUN.
```





# 7 Optimizing the Runtime Performance

---

This section describes several ways to enhance programs in order to obtain the best from your COBOL application.

The most important actions to be performed in order to optimize performance are:

- If available, use the `-server` Java option.
- Avoid debug information in programs.

By using debug compiler options, the runtime performance of the Jopaz compiled application is strongly affected. The runtime statistics are then no longer comparable with the COBOL runtimes.

Compiler options under `jopaz.compiler.options=` that cause a performance drop in this context are `-d` and `-dx`. For a more detailed description of what these two options do, please see section [“Compiler Options” on page 37](#).

- Avoid logging the Runtime activity.

By opening and closing the log file during program execution, significantly more workload is incurred. Therefore, if you are running a performance critical program, it is advisable to disable logging.

In this case, simply remove or comment out the following statements from your runtime definition (RUNOPT) or reduce the trace level:

- `jopaz.logfile=<path-to-trace>/jopaz.%u.txt`
- `jopaz.tracelevel=63`

For more information, see section [“Runtime Property Definition” on page 44](#).



# 8 Optimizing the Java Virtual Machine

---

■ Optimizing the Java Virtual Machine .....	92
---	----

## Optimizing the Java Virtual Machine

---

You can set many different options that change the behavior when starting or running the Java Virtual Machine (JVM). These options are optional, but some are set as default (hidden). There are also options that must be used to prevent security leaks.

The options are command-line options and can be set in the JVM configuration script. A subset of optimization options can be found below.

**Note:**

Not every option is recommended or applicable for each operating system.

These options are set in the configuration script or file of the JVM directly.

### Example for z/OS Java Optimzation

This example uses the option `-Xgcpolicy:optthruput` and `-Xquickstart`. It is applicable for all options described in this section. You can add options by separating them with blanks.

```
# Configure JVM options
# Add JVM optimization parameters here
# ☐=to continue in next line
IJO ="- Xquickstart -Xgcpolicy : optthruput "
# Uncomment the following to aid in debugging " Class Not Found " problems
IJO =" $IJO -verbose : class "
export IBM_JAVA_OPTIONS =" $IJO "
/*
```

## -X Comand-Line Options

These options are nonstandard. Typically, they are unique to a JVM implementation. These options are checked by the JVM. If not recognized, the JVM would not start.

For more information, check the Using -X command-line options section on the IBM Documentation website.

## -XX Comand-Line Options

These options are not checked for validity. These options are ignored if not recognized by JVM.

## Log4j Vulnerability Optimization

The following option is very important as it is significant for closing the Log4j vulnerability.

Option	Description
<code>-Dlog4j2.formatMsgNoLookups=true</code>	Disables inline message lookup that exposes environment variables and system configurations.

## External Calls

Option	Used for
<code>-Xgcpolicy:optavgpause</code>	Slightly increases unnormalized zIIP offload.
<code>-Xgcpolicy:optthruput</code>	Increases unnormalized zIIP offload.

## External Sort

Option	Used for
<code>-Xgcpolicy:optthruput</code>	Decreases the total GCP time, the job runtime, and the unnormalized zIIP offload the last of which has negative effect.

## Sequential Fixed-Blocked (FB) Files

### Note:

If two options are listed together, it is recommended that you use them simultaneously.

Option	Used for
<code>-Xgcpolicy:optavgpause</code> and <code>-Xshareclasses</code>	Slightly increase the unnormalized zIIP offload.  Slightly decrease the total GCP time and the job runtime.
<code>-Xgcpolicy:optthruput</code> and <code>-Xshareclasses</code>	Slightly increase the unnormalized zIIP offload.  Slightly decrease the total GCP time and the job runtime.
<code>-Xnoaot</code>	Slightly increase the unnormalized zIIP offload.  Slightly decrease the total GCP time and the job runtime.

## Sequential Variable-Blocked (VB) Files

Option	Used for
<code>-Xgcpolicy:optthruput</code>	Slightly decreases the total GCP time and the job runtime.

Option	Used for
-Xshareclasses	Slightly decreases the total GCP time and the job runtime.
-Xnoaot	Slightly decreases the total GCP time and the job runtime.

## VSAM Key Sequential Data Sets (KSDS)

Option	Used for
-Xgcpolicy:optthruput	Slightly decreases the total GCP time and the job runtime.
-Xgcpolicy:optavgpause	Slightly decreases the total GCP time and the job runtime.
-Xnojit	Increases the unnormalized zIIP offload. Slightly decreases the total GCP time and the job runtime.

## Default JVM Settings

Option	Parameter	Description
-Xaot		Ahead-of-Time compiler is enabled.  Allows the compilation of Java classes into native code for subsequent executions.  It is used for improving startup time of the same program.  Only active for classes found in shared classes cache.
-Xclassgc		Enables syanmix class unloading on demand.
-Xcompactexplicitgc		
-Xcompressedrefs		

Option	Parameter	Description
<code>-Xgcpolicy:parameter</code>	gencon	Controls which garbage collection policy is used for the Java application.  Gencon: best suited for transactional application with many short-lived objects. Minimizes the GC pause times without compromising throughput.
<code>-Xjit</code>		Signifies that Just-in-Time compiler is enabled.
<code>-Xlinenumbers</code>		
<code>-Xloa</code>		Enables the allocation of large object area (LOA) during garbage collection.  The allocations made in the small object area (SOA).  If there is not enough space and the object is larger than 64KB, the object is allocated in LOA.  If LOA is not used, size is shrunk to zero.
<code>-Xmaxe</code>		Expands settings.
<code>-Xsigcatch</code>		Activates VM signal handling code.





## 9 Troubleshooting and Statistics

---

■ Monitoring Statistics .....	99
■ String Index out of Bounds Exception .....	100
■ Activating Extended Tracing .....	101
■ Validating License File with LICUTIL .....	101
■ Increasing the Amount of Information in Error Messages .....	101
■ Setting the JCL TIME Parameter .....	102
■ Fixing Output Destination for COBOL Subprograms .....	102
■ Errors When Configuring Jopaz .....	102



# Monitoring Statistics

---

## Monitoring Statistics

Jopaz differs from other Software AG zIIP products in the way it evaluates statistics.

The time spent on the zIIP is usually evaluated as SRB time.

But Jopaz runs on the JVM and any Java workload runs on zIIP in TCB mode. Therefore, Java times are included in the JES job log time statistics reported by the IEF032I message.

### JES CPU Statistics

Here is an example for the JES CPU statistics from the IEF032I message:

```
IEF373I STEP / JAVAJVM / START 2023096.0808
IEF032I STEP/JAVAJVM / STOP 2023096.0808
          CPU: 0 HR 00 MIN 14.20 SEC SRB: 0 HR 00 MIN 00.01 SEC
```

From the statistics you can see a job step called JAVAJVM with a CPU time value of 14.20 seconds and an SRB time value of 0.01 seconds.

### Jopaz CPU Statistics

Jopaz also generates a CPU statistics report. Below you can find the corresponding Jopaz CPU statistics from the same run as the JES CPU Statistics:

```
STATISTICS FOR JOB RJPZM007 STEP JAVAJVM
GENERAL PROCESSORS: 5      ZIIPS: 1    NORMALIZ . FACTOR: 13.51
JOB STEP STARTED AT DATE TIME          06.04.2023 08:08:48.95
CURRENT DATE TIME                      06.04.2023 08:08:50.42
NUMBER OF EXCPS                        42860
TCB TIME ON GP (SECONDS)                0.648326
SRB TIME ON GP                         0.009572
TCB TIME ON ZIIP (NORMALIZED)          13.533963
ENCLAVE SRB TIME ON ZIIP ( NORMALIZED ) 0.000000
ENCLAVE SRB TIME ON GP                  0.000000
```

This Jopaz report is printed to either SYSOUT or to the JPZPRINT dataset optionally. From these statistics you can see the TCB times, which are summing up to 14.182 seconds. That makes the TCB time lower than the 14.20 secs from the respective JES2 IEF032I message.

### Understanding How Jopaz Calculates Statistics

The Jopaz TCB times are lower than the JES2 times because Jopaz starts counting when Java main gets control. Jopaz does not count the JVM start-up time.

The reason the JES2 CPU time is much higher than the total Jopaz GP time is because JES2 counts TCB time on zIIP as CPU time. The TCB workload is shifted to zIIP processors in Java scenarios. This cannot be seen from the JES IEF032I message.

You need to consider this fact when [“setting the JCL TIME parameter” on page 102.](#)

## String Index out of Bounds Exception

---

An error occurred while compiling the Cobol program with the Jopaz compiler:

```
java.lang.StringIndexOutOfBoundsException caught! (null)
-----
java.lang.StringIndexOutOfBoundsException caught! (null)
  in program HELWLD, paragraph _012_34_56_78_TEST_ALPNUM
(/opt/softwareag/jpz/jpz-compiler/sources/HELWLD.cbl:12)
(Top of stack)
  (java.lang.String.<init>(String.java:530))
  (com.jopaz.types.CobolVar.basicToString(CobolVar.java:246))
  (com.jopaz.types.CobolVar.toString(CobolVar.java:253))
  (com.jopaz.types.PicX.isNumeric(PicX.java:267))
```

with following Jopaz configuration:

```
JOPAZ JPZ-x.x.x.x_000
jopaz.compiler.options=-b -coe -cax -dznt -cv -cva -dcii -di -dv=0 -lf
-lo=/opt/softwareag/jpz/jpz-compiler/list
-jc -od=/opt/softwareag/jpz/jpz-compiler/output -pt2
-sp=/opt/softwareag/jpz/jpz-compiler/cpy -verobse
[Jopaz] jopaz.runtime.version=JPZ-x.x.x.x_000
```

where *JPZ-x.x.x.x\_000* is your Jopaz release version,

1. Enter the following code:

```
jopaz.array_check=1 // Array boundaries are checked at runtime=more details
                      //in case of "out of bounds" errors .
jopaz.display_message =1 //All messages are sent to SYSOUT .
```

2. If the following error occurs during the execution with the previously mentioned options, try to use the option `-m1` in addition. For this, the comments on the `-m1` definition must be observed:

```
Index out of bound CHECK ARRAY : detected index out of bound W- ASBORIND (0) [19]
```

```
-m1 // -Could prevent the IndexOutOfBoundsException.
// -The -m1 put all of WORKING-STORAGE into a contiguous block of memory.
// -It allow to run correctly in case of memory violation like for example to use
// index of occurs outside min or max limits.
// -In this case, occurs could "violate" memory programmatically accessing
// upper or lower working storage.
// -Using the -m1 you have to comment out or remove the jopaz.array_check =1
```

3. If the `StringIndexOutOfBoundsException` still exists when using the compiler option `-m1`, add the following option to your configuration:

```
jopaz.exception.java =1 // True = Internal java methods are traced in exception
messages
// False = COBOL paragraph names are traced in exception messages
// Default value is false
```

## Activating Extended Tracing

You can trace information about program starts and program ends, paragraph starts and paragraph ends, and the file I/O activities in the logfile with the following option:

```
jopaz.tracelevel=15
```

Additionally, you must add in the cobol source code the following routine:

```
C$WRITELOG
```

## Validating License File with LICUTIL

The Jopaz installation set also includes a JCL job example to run the licence validation utility (LICUTIL) of the Software AG Mainframe Licence Check (MLC). You can find LICUTIL in the Jopaz JOBS library after installation. The supplied job example only serves the purpose of checking the licence and outputting its components. For more information and functionalities of LICUTIL please check the Software AG MLC product documentation.

```
/*-----
/* Section to set parameters for job
/*-----
/* SET JPZLIC=JPZ_LICENSE_FILE
/* SET PRODVER=JPZBT1.1
```

In the JCL Symbolic Parameter Section, the DSN of the licence file (JPZLIC) is filled directly with the value from the `jpz_variabale_input_file.txt` during installation. You must only check the value for PRODVER and change it if necessary. This is made up of the product code and product version separated by a dot.

### Note:

You cannot check special license codes from the range from JPZ00 to JPZ99, because LICUTIL expects the product to be 3 or 5 alpha characters. The format of the PRODVER parameter is `PPP[PP]V.R[.S]`. When you try to check JPZ01 for example, this produces an incorrect product code error, because it expects the product code tag to be JPZ, not JPZ01.

## Increasing the Amount of Information in Error Messages

In case of unclear errors, you can set this option to get more information.

```
jopaz.exception.java=0 //True = Internal java methods are traced in exception messages
//False = COBOL paragraph names are traced in exception messages
//Default value is false
```

If the option `jopaz.runtime.native.dynamic.ignore_errors=0` is set, it is recommended to set `jopaz.exception.java=1`.

## Setting the JCL TIME Parameter

---

The JCL TIME parameter counts the TCB time on GP, the SRB time on GP, and the TCB time on zIIP. The SRB time on zIIP is ignored by the JCL TIME parameter.

When you are setting the JCL TIME parameter, you must consider all of the times the parameter counts. This means you might need to increase the JCL TIME parameter.

## Fixing Output Destination for COBOL Subprograms

---

The JZOS Batch Launcher uses the `STDOUT` DD name for the Java `System.out` stream. COBOL uses `SYSOUT` as the default DD name for the `DISPLAY` output.

When a Jopaz compiled main program calls a COBOL-compiled subprogram, the output from the Jopaz main program and the output from the COBOL subprogram go to different output devices.

You can recompile the COBOL subprogram with the `OUTDD(STDOUT)` option, and the `DISPLAY` output is written to the same destination as the Java `System.out` stream.

## Errors When Configuring Jopaz

---

### Invalid/Unsupported File Name Assignment in Source Code

If you receive this error

```
DynamicMVSSequential table:, oper=, RETCODE=47, 4, EDC5047I An invalid file name was
specified as a function parameter.
```

it means that some file assignments contain prefixes like `DA-S-` which prevent the class to execute without an error.

For example:

```
000420 SELECT XXXXXXXX ASSIGN TO DA -S- XXXXXXXX . TFI301Y
000430 SELECT XXXXXXXX ASSIGN TO DA -S- XXXXXXXX . TFI301Y
000440 SELECT XXXXXXXX ASSIGN TO DA -S- XXXXXXXX . TFI301Y
```

### Solution

You can use the option `-csl`. It treats the COBOL name `ASSIGN` clause as a literal. This allows a mapping to be created for the file name if the `jopaz.file.env_naming` (boolean) compile property is set to true in the `COMOPT` file.

#### Note:

This option is in conflict with `-cax`, which specifies the default file assignment as external.

## Passing Parameters from JCL Using PARM to Linkage Section in Cobol

Passing parameters through ARGS is working fine besides when setting a value directly or when using a variable in the JCL. When using PARM in the substitution step of JVM EXEC, the value is not set.

### Solution

The fully qualified main class name and any arguments need to be specified as PARM= strings to the batch launcher program.

```
//* JAVACLS =& CLASS ,
/* PARM =& PARA
// PARM ='& CLASS & PARA '
```

## Calculation of FNR and DbID for Adabas Fails

The IBM compiler option TRUNC(STD) is set.

### Solution

Add the Jopaz compiler option -dznt.

## Error Solutions

This section provides detailed description to several errors as well as possible ways to resolve them.

Error	Explanation	Option
#144 Invalid name	Possibly one of the selected variable names is a reserved word.	-rw=word
#27 Invalid level number	This is a follow-up error if an error occurs in the line before it.	
#15 Unexpected token	This can be a follow-up error like #27 Invalid level number or can mean a reserved word is used.	

