

|  |  |
| --- | --- |
| An introduction and overview Autosys |  |
| With its multi-platform support, AutoSys Workload Automation is an automated job control system that excels in scheduling, monitoring, and reporting AutoSys jobs, making it essential for enterprise applications.  By Randall Fadler September 2024 |  |

Table of Contents

[***An introduction and overview*** 1](#_Toc177040845)

[1. Introduction 3](#_Toc177040846)

[Overview of Autosys 3](#_Toc177040847)

[Importance of JIL (Job Information Language) 3](#_Toc177040848)

[2. Getting Started with JIL 3](#_Toc177040849)

[What is JIL? 3](#_Toc177040850)

[Basic Syntax and Structure 4](#_Toc177040851)

[Creating Your First JIL File 4](#_Toc177040852)

[3. JIL Subcommands 4](#_Toc177040853)

[Overview of Subcommands 4](#_Toc177040854)

[4. JIL Attributes 4](#_Toc177040855)

[Detailed Explanation of JIL Attributes 4](#_Toc177040856)

[Examples of Attributes for Different Job Types 4](#_Toc177040857)

[5. Common JIL Parameters 5](#_Toc177040858)

[You can find more detailed information and additional parameters in the Broadcom AutoSys documentation1. 8](#_Toc177040859)

[6. Advanced JIL Features 8](#_Toc177040860)

[Box Jobs and Their Attributes 8](#_Toc177040861)

[Box Job and Job Dependencies and Conditions 8](#_Toc177040862)

[Calendars and Schedules 9](#_Toc177040863)

[7. Loading JIL Files 12](#_Toc177040864)

[How to Load JIL Files into the Autosys scheduler 12](#_Toc177040865)

[Verifying and Troubleshooting JIL Scripts 12](#_Toc177040866)

[8. Real-World Examples 12](#_Toc177040867)

[9. Best Practices 14](#_Toc177040868)

[Tips for Writing Efficient and Maintainable JIL Scripts 14](#_Toc177040869)

[Common Pitfalls and How to Avoid Them 14](#_Toc177040870)

[10. Resources 14](#_Toc177040871)

[Links to Official Documentation and Further Reading 14](#_Toc177040872)

# 1. Introduction

## Overview of Autosys

Previously known as CA Workload Automation, Autosys is an automated job control system for scheduling, monitoring, and reporting. It provides a single point of control for managing and visualizing complex job workflows across multiple platforms.

AutoSys Workload Automation is a distinguished multi-platform automated job control system.

* It provides robust capabilities for scheduling, monitoring, and reporting AutoSys jobs. The text-based reports provides visual information concerning the state of the job. For example, Running, Inactive, Failed, etc.
* Job definitions and JIL (Job Information Language) are managed on a centralized AutoSys server inside an Oracle database. The JIL definitions are loaded into the Autosys server through the JIL command.
* The AutoSys server communicates with remote AutoSys agents installed on machines like Windows and Unix to execute job operations. Many different types of commands can be used. For example, you can execute a Python script, execute a direct SQL statement, or initial a wait job which looks for a file before running to success.
* Supports databases like Microsoft SQL database, Oracle database, Sybase, and IBM DB2.

## Importance of JIL (Job Information Language)

JIL is the scripting language used to define and manage jobs in Autosys. It allows users to create, update, and delete job definitions, making it a crucial tool for job scheduling and automation.

# 2. Getting Started with JIL

## What is JIL?

JIL (Job Information Language) is a scripting language used to define job attributes and dependencies in Autosys. It provides a way to describe the characteristics and behavior of jobs.

## Basic Syntax and Structure

A JIL script consists of a series of job definitions, each starting with the insert\_job subcommand followed by job attributes. Here's an example:

insert\_job: my\_first\_job

command: echo "Hello, World!"

machine: localhost

owner: user

start\_times: "12:00"

## Creating Your First JIL File

To create a JIL file, open a text editor and write your job definitions. Save the file with a .jil extension.

# 3. JIL Subcommands

## Overview of Subcommands

Subcommands in JIL are used to perform actions such as inserting, updating, or deleting jobs. Common subcommands include insert\_job, delete\_job, and update\_job.

#### Examples of Common Subcommands

* insert\_job: Defines a new job.
* delete\_job: Removes an existing job.
* update\_job: Modifies an existing job.

# 4. JIL Attributes

## Detailed Explanation of JIL Attributes

JIL attributes define the properties and behavior of jobs. Some common attributes include:

* command: The command to be executed.
* machine: The machine where the job will run.
* owner: The user who owns the job.
* start\_times: The time when the job should start.

## Examples of Attributes for Different Job Types

* Command Job:
* Box Job:

# 5. Common JIL Parameters

- insert\_job

- Description: Defines a new job.

- Example: insert\_job: my\_job

- job\_type

- Description: Specifies the type of job (e.g., CMD, BOX, FW, FT, SQL).

- Example: job\_type: CMD

- command

- Description: The command to be executed by the job.

- Example: command: /path/to/script.sh

- machine

- Description: The machine where the job will run.

- Example: machine: server1

- owner

- Description: The user who owns the job.

- Example: owner: user

- start\_times

- Description: The time(s) when the job should start.

- Example: start\_times: "12:00"

- condition

- Description: Specifies job dependencies.

- Example: condition: success(job\_a)

- box\_name

- Description: The name of the box job that contains this job.

- Example: box\_name: my\_box

- description

- Description: A description of the job.

- Example: description: "This is a sample job"

- run\_calendar

- Description: The calendar that defines the job's schedule.

- Example: run\_calendar: my\_calendar

- exclude\_calendar

- Description: The calendar that defines the dates to exclude from the job's schedule.

- Example: exclude\_calendar: holiday\_calendar

- std\_out\_file

- Description: The file where the job's standard output will be written.

- Example: std\_out\_file: /path/to/stdout.log

- std\_err\_file

- Description: The file where the job's standard error output will be written.

- Example: std\_err\_file: /path/to/stderr.log

- alarm\_if\_fail

- Description: Specifies whether to raise an alarm if the job fails.

- Example: alarm\_if\_fail: 1

- max\_run\_alarm

- Description: The maximum run time before raising an alarm.

- Example: max\_run\_alarm: 60

- min\_run\_alarm

- Description: The minimum run time before raising an alarm.

- Example: min\_run\_alarm: 5

- profile

- Description: The profile to be used for the job.

- Example: profile: /path/to/profile

- permission

- Description: The permissions for the job.

- Example: permission: gx,ge

- date\_conditions

- Description: Specifies whether the job has date conditions.

- Example: date\_conditions: 1

- days\_of\_week

- Description: The days of the week when the job should run.

- Example: days\_of\_week: mo,tu,we,th,fr

- start\_mins

- Description: The minutes past the hour when the job should start.

- Example: start\_mins: 15

- term\_run\_time

- Description: The maximum run time before terminating the job.

- Example: term\_run\_time: 120

- box\_terminator

- Description: Specifies whether the job should terminate the box if it fails.

- Example: box\_terminator: 1

- job\_terminator

- Description: Specifies whether the job should terminate if a specified job fails.

- Example: job\_terminator: job\_b

- priority

- Description: The priority of the job.

- Example: priority: 10

# You can find more detailed information and additional parameters in the [Broadcom AutoSys documentation1](https://techdocs.broadcom.com/us/en/ca-enterprise-software/intelligent-automation/autosys-workload-automation/12-0-01/reference/ae-job-information-language.html).

# 6. Advanced JIL Features

## Box Jobs and Their Attributes

Box jobs are containers for other jobs, allowing you to group related jobs together. Attributes for box jobs include box\_name and box\_terminator.

## Box Job and Job Dependencies and Conditions

Dependencies define the relationships between jobs. You can use attributes like condition to specify job dependencies. For example:

insert\_job: my\_box

job\_type: BOX

owner: user

machine: server1

description: "This is a box job containing three jobs"

start\_times: "12:00"

insert\_job: job1

box\_name: my\_box

command: /path/to/script1.sh

machine: server1

owner: user

description: "This is the first job inside the box"

start\_times: "12:05"

insert\_job: job2

box\_name: my\_box

command: /path/to/script2.sh

machine: server1

owner: user

description: "This is the second job inside the box"

condition: success(job1)

start\_times: "12:10"

insert\_job: job3

box\_name: my\_box

command: /path/to/script3.sh

machine: server1

owner: user

description: "This is the third job inside the box"

condition: success(job2)

start\_times: "12:15"

Explanation:

* Box Job (my\_box): This is the container job that holds the other jobs. It starts at 12:00.
* Job 1 (job1): This job runs the script script1.sh and starts at 12:05.
* Job 2 (job2): This job runs the script script2.sh and starts at 12:10, but only if job1 is successful.
* Job 3 (job3): This job runs the script script3.sh and starts at 12:15, but only if job2 is successful.

This setup ensures that the jobs inside the box run in a specific sequence based on their dependencies.

## Calendars and Schedules

Calendars and schedules allow you to define when jobs should run. You can use attributes like run\_calendar and exclude\_calendar to specify job schedules.

**Example 1: Using**run\_calendar

This example shows a job that runs according to a specific calendar.

insert\_job: job\_with\_run\_calendar

job\_type: CMD

command: /path/to/script.sh

machine: server1

owner: user

description: "This job runs according to the specified run calendar"

run\_calendar: my\_run\_calendar

start\_times: "12:00"

**Explanation:**

* **run\_calendar**: Specifies the calendar that defines the job’s schedule. The job will run on the dates defined in my\_run\_calendar.

**Example 2: Using**exclude\_calendar

This example shows a job that runs every day except for the dates specified in the exclude calendar.

insert\_job: job\_with\_exclude\_calendar

job\_type: CMD

command: /path/to/another\_script.sh

machine: server1

owner: user

description: "This job runs every day except for the dates in the exclude calendar"

run\_calendar: all\_days

exclude\_calendar: holiday\_calendar

start\_times: "12:00"

**Explanation:**

* **run\_calendar**: Specifies the calendar that defines the job’s schedule. In this case, all\_days means the job is scheduled to run every day.
* **exclude\_calendar**: Specifies the calendar that defines the dates to exclude from the job’s schedule. The job will not run on the dates defined in holiday\_calendar.

These examples demonstrate how to use the run\_calendar and exclude\_calendar

In AutoSys, **run calendars** are used to define specific dates or criteria for scheduling jobs. They allow you to customize job schedules based on your organization’s needs. Here are the key types of calendars you can use:

**Types of Calendars**

1. **Standard Calendar**
   * **Description**: Specifies a list of fixed dates, such as holidays.
   * **Usage**: Useful for scheduling jobs on specific dates.
   * **Example**:
   * calendar: holiday\_calendar
   * 01/01/2024
   * 12/25/2024
2. **Cycle**
   * **Description**: Specifies a list of date ranges (periods).
   * **Usage**: Useful for scheduling jobs over recurring periods.
   * **Example**:
   * cycle: quarterly\_cycle
   * start\_date: 01/01/2024
   * end\_date: 03/31/2024
   * start\_date: 04/01/2024
   * end\_date: 06/30/2024
3. **Extended Calendar**
   * **Description**: Specifies a set of criteria that the utility uses to generate a list of fixed dates. It might reference a cycle and apply rules to the periods in it.
   * **Usage**: Useful for more complex scheduling needs.
   * **Example**:
   * ext\_calendar: fiscal\_calendar
   * workday: X X X X X . .
   * non\_workday: O S N W P

**Managing Calendars**

[You can manage calendars using the autocal\_asc command, which provides a text-based command-line mechanism to define, modify, and delete calendar definitions1](https://techdocs.broadcom.com/us/en/ca-enterprise-software/intelligent-automation/autosys-workload-automation/12-1-01/reference/ae-commands/define-workload-objects/autocal-asc-command-manage-calendars.html). Here are some common tasks:

* **Create a Calendar**: Define a new calendar with specific dates or criteria.
* **Modify a Calendar**: Update an existing calendar to add or remove dates.
* **Delete a Calendar**: Remove a calendar from the system.

**Example Usage**

**Creating a Standard Calendar**

autocal\_asc -c holiday\_calendar

01/01/2024

12/25/2024

**Creating an Extended Calendar**

autocal\_asc -e fiscal\_calendar

workday: X X X X X . .

non\_workday: O S N W P

Run calendars are essential for customizing job schedules and ensuring that jobs run according to your organization’s specific requirements.

# 7. Loading JIL Files

## How to Load JIL Files into the Autosys scheduler

To run a JIL file, use the jil command followed by the file name:

jil < my\_first\_job.jil

The jil command is used to interact with the Autosys database. It allows you to insert, update, and delete job definitions.

## Verifying and Troubleshooting JIL Scripts

After running a JIL file, you can use the autorep command to verify job definitions and check for errors:

autorep -j my\_first\_job

# 8. Real-World Examples

1. Data Warehousing and ETL Processes

AutoSys can schedule and manage ETL (Extract, Transform, Load) jobs, ensuring that data is extracted from source systems, transformed according to business rules, and loaded into data warehouses efficiently. This is crucial for maintaining up-to-date and accurate data for business intelligence and analytics.

2. Batch Processing

In financial institutions, AutoSys is used to automate batch processing jobs, such as end-of-day processing, report generation, and data reconciliation. This ensures that critical financial data is processed accurately and on time.

3. Application Integration

AutoSys can integrate and manage jobs across different applications, such as ERP systems, CRM systems, and custom applications. This helps in coordinating workflows that span multiple systems, ensuring data consistency and process synchronization.

4. Cloud Automation

AutoSys is essential for managing jobs in cloud environments. It can automate workflows that involve cloud services, such as data migration, backup, and resource provisioning. [This reduces manual errors and improves efficiency in cloud operations1](https://community.broadcom.com/enterprisesoftware/blogs/michael-woods1/2021/04/08/why-autosys-is-so-important-in-todays-cloud-first)[2](https://academy.broadcom.com/blog/automation/why-autosys-is-so-important-for-cloud-automation).

5. IT Operations and Maintenance

AutoSys is used to automate routine IT operations and maintenance tasks, such as system backups, software updates, and log file management. This helps in maintaining system health and reducing downtime.

6. Retail and Supply Chain Management

In retail, AutoSys can automate inventory management, order processing, and supply chain workflows. This ensures that products are available when needed and that orders are processed efficiently.

7. Healthcare and Life Sciences

AutoSys can manage and automate workflows in healthcare, such as patient data processing, lab result integration, and compliance reporting. This helps in improving patient care and ensuring regulatory compliance.

8. Telecommunications

In the telecommunications industry, AutoSys is used to automate network management tasks, such as monitoring network performance, managing configurations, and handling alerts. This ensures reliable and efficient network operations.

# 9. Best Practices

## Tips for Writing Efficient and Maintainable JIL Scripts

* Use meaningful job names and descriptions.
* Group related jobs into box jobs.
* Define clear job dependencies and conditions.
* Regularly review and update job definitions.

## Common Pitfalls and How to Avoid Them

* Avoid hardcoding paths and parameters.
* Test JIL scripts in a development environment before deploying to production.
* Monitor job execution and handle errors gracefully.

# 10. Resources

## Links to Official Documentation and Further Reading

* [Broadcom Autosys Documentation](https://techdocs.broadcom.com/us/en/ca-enterprise-software/intelligent-automation/autosys-workload-automation/12-0-01/reference/ae-job-information-language.html)
* [Autosys JIL Reference Guide](https://docs.autosys.com/jil-reference-guide)