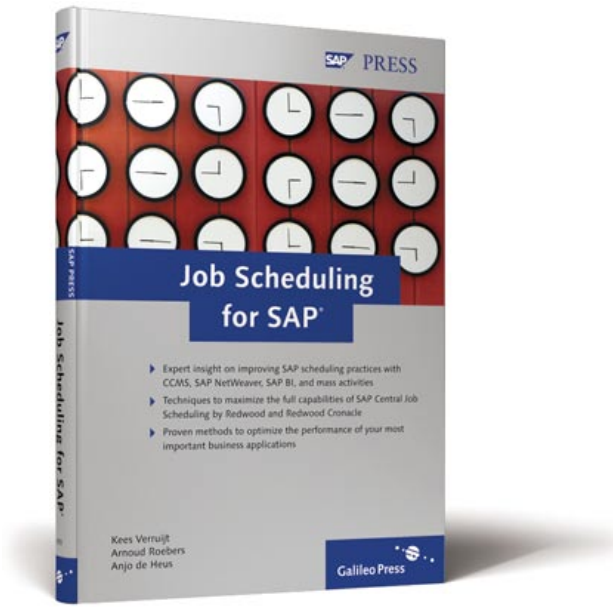
Kees Verruijt, Arnoud Roebers, Anjo de Heus

Job Scheduling for SAP®



SAP PRESS

Contents at a Glance

Contents

Foreword

IT Process automation enables organizations to efficiently manage the increasingly complex technical requirements demanded by real-time business processes. Companies can achieve maximum productivity, agility, and resource utilization through intelligent, centralized management of application workload across all environments. Event-driven automation ensures immediate and appropriate responses to business requirements, enabling the enterprise to take a dynamic approach in managing unpredictable workloads and mitigating risk, assuming that it is built to be both robust and reliable.

SAP and Redwood Software have been collaborating to integrate Redwood's job-scheduling technology into SAP NetWeaver. Together, the companies offer an adaptive, scalable application that provides the real-time, event-driven job scheduling and process automation needed to manage background processing across multiple systems and applications. Through a central point of control, it enables organizations to orchestrate automation of the IT workload generated by modern business practices and to manage efficiently the enterprise applications this orchestration requires. With its comprehensive automation rules, it manages both routine and predictable workload as well as the ad hoc tasks that can be generated at any time by applications such as enterprise resource planning (ERP), business-to-business processes, and self-service scenarios.

The job-scheduling application treats both planned and unplanned workloads as "events" and builds relationships between these events to create dependencies where appropriate. By detecting and responding to business events, it enables a dynamic approach to workload process automation. And because it responds immediately to events, it provides the business agility, throughput, and efficiency needed to optimize the use of available time and resources. Processing that is balanced across all available time instead of being restricted by artificial date and time boundaries helps to avoid bottlenecks, keep user responses within agreed-upon service levels, and maximize the use of computing resources. As a result, it increases return on investment (ROI) and makes it possible to reduce total cost of ownership (TCO), thus deferring the need to acquire additional hardware.

With SAP Central Job Scheduling by Redwood, customers are leveraging the combined expertise of both SAP and Redwood Software, gained through

serving more than 30,000 customers worldwide in more than 150 countries. Both companies have widespread experience with organizations of all sizes, not just large enterprises. And SAP has decades of experience helping organizations improve their business fundamentals, business processes, and overall business results. Implementing the job-scheduling application initiates real-time, event-driven job-scheduling and process-automation functionality that will help to optimize business processes and meet constantly changing customer and market requirements.

Enjoy this book as it leads you to insights and ideas about how best to run your operative IT processes, not just for SAP landscapes but far beyond for your whole IT environment.

Walldorf, June 2006
Klaus Kreplin
Executive Vice President SAP NetWeaver, SAP AG

Preface

This book helps you learn to automate processes in your IT systems and optimize the performance of your most important business applications, using the appropriate job-scheduling functionality supported by SAP NetWeaver.

Job scheduling in SAP environments is changing rapidly. Although SAP R/3 had job-scheduling capabilities built in, and SAP BI and other solutions extended these, we will explain why these capabilities are insufficient for the sort of solution envisioned and proposed with SAP NetWeaver and the Enterprise SOA—also known as enterprise services architecture (ESA). These require asynchronous processing of business processes.

Job scheduling and asynchronous service processing are closely related. Scheduling is concerned with starting tasks at the appropriate time, determined by rules and external conditions. Asynchronous processing allows a task to be decoupled from its initiation. Schedulers can provide asynchronous processing facilities by allowing systems to request execution of a task at a particular time (or on a particular schedule), or when triggered by an external event.

Traditionally, job-scheduling environments were driven by the following needs:

- Automated operations
- High-volume (overnight) operating system jobs
- Schedule workload based on clocks and calendars
- Inter-job dependencies

These requirements are met by any scheduling system that supports rules-based calendaring and dependencies. We will describe the origins of such job schedulers in **Chapter 1**, which covers non-SAP job scheduling, and **Chapter 2**, which covers decentralized SAP job scheduling.

Modern scheduling systems need to deal with more complex requirements. They need to:

- Provide a single point of control
- Help IT alignment with business processes
- Provide event-driven processing
- Integrate with packaged applications (ERP, CRM)

- Manage workload and dependencies across heterogeneous platforms and applications (UNIX, NT, z/OS, AS/400, J2EE)
- Provide asynchronous application integration between disparate systems
- Provide asynchronous application integration between new technologies such as Web services.

Modern scheduling systems are asynchronous and event-based. Clock and calendar triggers are still important, but only when this is a business requirement. Triggers based on events such as file arrival, message-oriented middleware, and complex application triggers are becoming more important. SAP realized this and created interfaces for all its major components so that external job schedulers can work with the SAP software applications. These interfaces are described in **Chapter 3**. We will describe the requirements for such central job schedulers in **Chapter 4**.

With SAP NetWeaver 2004, SAP has started to offer a basic and a full version of *SAP Central Job Scheduling by Redwood*: the solution for complete job-process management in SAP oriented environments. This centralized job scheduler is based on Redwood's Cronacle *for SAP solutions*. We will explain the differences are between the offered versions and examine the components that make up the different versions in **Chapter 5**.

In the next three chapters we give you practical advice. **Chapter 6** gives insight into the installation and deployment process. **Chapter 7** covers the objects, principles, and processes that Cronacle uses. This chapter will explain how Cronacle satisfies the requirements set forth in Chapter 4. **Chapter 8** covers operations.

We conclude in **Chapter 9** by studying three business cases, including references to actual companies who have already implemented the vast job-scheduling functionality.

Acknowledgements

There are many people who we would like to thank for contributing to this book. We know that many people contributed, both directly and indirectly, often in their free time, during evenings and nights as well as on their weekends. We would like to thank Klaus Kreplin for writing such a great preface to this book. We really appreciate his time and belief. We would like to thank our colleagues at Redwood Software for contributing to the success of

Redwood Software, without whom both Cronacle and this book never would have existed.

We would like to thank Astrid Tschense at SAP for her help on Java and Java scheduling. We would also like to thank our customers, especially LVR, Actebis, and Freudenberg: not only for implementing Cronacle long before it became the standard, but also for providing us with the great practical examples used in this book. We're sure many readers will find these stories provide interesting insights.

We would like to thank Anton Goselink, Chris Lewis, Chris Lentz, Joe Kuncharia and Joachim Weide at Redwood Software as well as Christoph Nake and Melanie Reinwarth at SAP AG for proofreading this book, making it more readable, and for their many suggestions and tips for improving the book. To Wim Jansen and Hans Plasmeijer: Thanks for the interesting and useful conversations we had about Cronacle, development customer experience, etc. Thanks also to Francis Dropik, who created some of the graphics used throughout this book.

From Deloitte consultancy we would like to thank Marcel Kuijper and Rob de Maat for much more of their valuable time than they probably imagined they signed up for. We also thank Anke de Jongh for providing the readers perspective from a non-technical view.

We would like to, specifically, thank Dieter Babutzka at SAP for sharing his experiences with SAP scheduling and CCMS. The many discussions we had with him proved invaluable for the insights we now have. We thank Jörg Heinemann at Actebis and Peter Oswald at Freudenberg IT KG for their extensive examples used in this book.

Special thanks go to Andrew Evers at Redwood Software for his contribution on Java scheduling. He knows this topic inside and out. For a Java developer, he turned out to be quite a writer. A special thanks also to Ed Wenmakers at Redwood Software for his ongoing assistance in all the little technical SAP details. All his knowledge on scheduling and SAP interfaces and particularly SAP internals proved to be very helpful while writing this book. Another special thanks to Edwin Esser at Redwood Software. Both his contributions and wrapping together all the chapters were a great help to us.

Finally, we would like to thank Stefan Proksch, Florian Zimniak, and the team at Galileo Press for making this book possible in the first place.

And, since we sometimes forget to mention every important person, a collective "thank you all".

Houten, June 2006
Kees Verruijt
Arnoud Roebers
Anjo de Heus

This chapter will give you an overview of the principles and processes of the SAP Central Job Scheduling by Redwood solution. The principles are mainly concerned with the different object types that SAP Central Job Scheduling by Redwood supports.

7 Principles and Processes

Within SAP Central Job Scheduling by Redwood, every object type represents a particular construct, such as a job definition, job, or queue. All objects of all object types together form the customization of the SAP Central Job Scheduling by Redwood system to your specific needs. The principles and processes of the basic and full versions of SAP Central Job Scheduling by Redwood are the same.

SAP Central Job Scheduling by Redwood uses an extensive list of different object types to satisfy all requirements. The object types can be seen in Figure 7.1, which shows the expanded explorer tree from Redwood Explorer.

Not all object types are required in all situations. We will cover the key principles that most if not all customers will use, or should consider using. These are:

- User management and security
- Job definitions
- Job allocation
- Job management
- Job submission
- Output management
- SAP job management

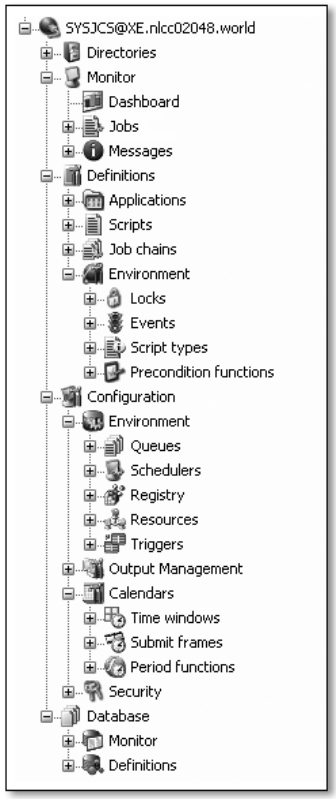


Figure 7.1 All Objects of SAP Central Job Scheduling by Redwood

7.1 User Management and Security

User management, and in particular granted security privileges, is an important part of the job-scheduling solution. As we will see, the product supports fine-grained privileges, accommodating any setup required.

7.1.1 User Accounts

For historical reasons, users of SAP Central Job Scheduling by Redwood are also Oracle users. For every scheduler user that needs to log in, an Oracle user account with the same name needs to exist. SAP Central Job Scheduling by Redwood (full version) has an optional plug-in that can synchronize the Oracle users with an LDAP server. Users can log in using their LDAP credentials; if the LDAP server authenticates these, then an Oracle and SAP Central Job Scheduling by Redwood account is created with the appropriate privileges.

The next release of SAP Central Job Scheduling by Redwood (basic version) will be embedded in SAP NetWeaver, and thus will support UME for user management.

7.1.2 Roles and Privileges

What all versions share is an internal model that uses *roles* to ease privilege maintenance. Privileges such as access to a particular object or action can be granted directly to a user, or to a role. Roles in turn are granted to other roles or users. This explicitly allows multiple levels of roles. A special built-in role called PUBLIC means access to everyone is allowed.

Many privileges can be granted in SAP Central Job Scheduling by Redwood. These are not granted to a role or user, but they grant something on a particular group of objects. They can grant something on just a single object, on all objects of a particular type in a partition, or on all objects in the entire repository.

Once you have created roles and privileges, the Redwood Explorer user interface allows you to create predefined function profiles that combine a set of roles, account settings, and privileges. This allows you to re-use function profiles when new users are created, as shown in Figure 7.2.

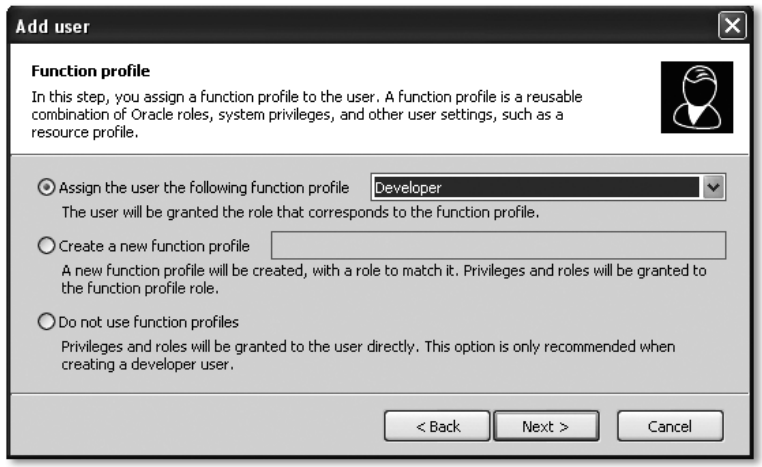


Figure 7.2 Function Profile

For ease of use, the next release of SAP Central Job Scheduling by Redwood (basic version) will have predefined roles that are not modifiable by the customer.

7.1.3 Partitions and User Classes

A partition is a separated logical unit of authorization that is introduced with Release 7.0 of SAP Central Job Scheduling by Redwood (full version). Partitions are used to group scheduler objects: Their namespaces include partition names, and objects live inside partitions. For example, the queue `SYSTEM` from Version 6 is known as queue `SYSJCS.SYSTEM` in the next release.

Objects in *shareable partitions* can be shared with other partitions. Objects in *private partitions* hide their objects from the rest of the system. Private partitions can be used to completely separate different customers, applications, or other business subdivisions.

Partitions are repository objects that can be maintained like any other repository objects. They are also granted to users and roles. When users have access to one or more partitions, they can choose the partitions that they want to work with at any time without changing the connection.

Most repository objects are partitioned. The object types that are not partitioned are jobs, users, roles, user-classes, time zones, and partition objects themselves. Even if users are not partitioned, groups of users can be hidden from each other. Every Redwood user can be assigned to a user class. If this is assigned, the user will only be able to see users with the same user class. This in itself provides a kind of partitioning.

When we compare the segmentation of partitions with the client concept available in SAP, you can see that it is possible to implement clients using partitions but not the other way around. Users are able to enable multiple partitions at the same time, whereas with clients only one can be used at a time. Furthermore, clients only form a division for transactional and master data and for customizing settings, but not for program code.

Shareable partitions are generally used to store objects that are shared with other partitions. The `SYSJCS` and `RSI` partitions that are delivered with the system are good examples of this. These partitions contain objects that can be reused (referred to) by customer applications.

Private partitions are used to store objects such as job definitions that must remain completely private to the users of that private partition. As such they are generally used to model a particular customer. Private partitions will not be available in all versions of Cronacle or SAP Central Job Scheduling by Redwood.

When you log in to the Redwood Explorer and find there are multiple partitions, the system will allow you to choose which partitions to work with during the session. You can change this set at any time using the drop-down on the environment bar, as shown in Figure 7.3.

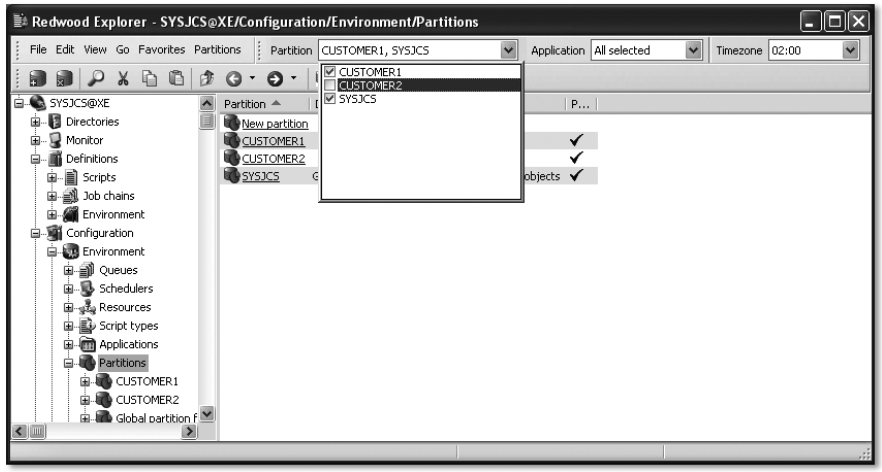


Figure 7.3 Choosing the Enabled Partitions

Example

An *application service provider* (ASP) has a range of customers. The ASP does not want customers to know what other customers they have, and customers don't want to risk having other customers use the objects that they create. Creating shareable partitions containing job definitions that the ASP provides to all customers can solve this scenario. For every customer, a partition is created that will store the objects for that customer, as well as a user class. The employees of that customer are all assigned to the user class.

7.2 Job Definitions

SAP Central Job Scheduling by Redwood makes a clear distinction between job definitions and their executions, called *jobs*. Job definitions must be predefined before a job can be run, so in this sense SAP Central Job Scheduling by Redwood is much stricter than other job schedulers.

By insisting that you think first before you act, SAP Central Job Scheduling by Redwood makes sure that you can re-use job definitions, in two ways: by repeated executions, and by copying master job definitions and then modi-

fying the copy. This maximizes re-use and prevents accidental mistakes in the command that is executed ("Oops, tonight we tried to run `make_bakup` instead of `make_backup` ...") as well as in the parameters passed ("Oops, I meant `Friday`, not `Fridaj` ...").

A job definition is the schedulable unit of work. From the viewpoint of the user who submits a job definition, it does not matter what the content of the job definition is. He or she fills in job definition specific parameters, general scheduling parameters, and general job allocation parameters.

There are two distinct classes of job definitions: *scripts*, which perform a single block of code, and *job chains,* which run a number of subtasks in the form of other job definitions. This can be called the implementation of the task. All other parts of the job definition are the same in scripts as they are in job chains.

7.2.1 Job Definitions' Common Functionality

Common to both coded scripts and job chains are many attributes and settings. The most commonly used attributes define job allocation and management, such as a mandatory queue where the job must run, and the priority this particular job definition has when the system has to choose between multiple jobs.

The other part of a job definition is that it defines which job specific *parameters* can be passed to it, as well as which values are allowed. Parameters have a data type, which influences the way that the user interface allows the user to type in values. For `DATE` or `TIMESTAMP` parameters, for instance, it will produce a date picker where dates can easily be entered. *Constraints* define which values are allowed. They can refer to database tables, to fixed lists, or to expressions that calculate whether the entered value is correct.

Parameters and constraints are used frequently in the SAP Central Job Scheduling by Redwood system. All SAP job specific attributes are translated into parameters. This allows an end user who submits the job to say on which SAP system and client the SAP job should be run. Values are then checked against a table containing the known SAP systems, clients, ABAP program names, etc.

Job definitions can also specify that the job cannot run unless a specific event has been raised, or that a specific event will be raised when the job reaches a particular status.

7.2.2 Scripts

Scripts contain an implementation that consists of operating system or database code. OS commands can be written as an expression that is evaluated into an OS command for the shell that the user normally uses, or in an explicit command language such as KSH, CMD (Microsoft Windows only), or Perl. Cronacle also has special built-in capabilities for sending JES jobs to IBM z/OS mainframes and Java jobs to Java process servers.

Note that the basic versions of SAP Central Job Scheduling by Redwood will not allow you to run scripts other than those implementing the Cronacle *for SAP solutions* interface.

Parameters and variables

Cronacle uses parameters in the job definition to specify which variable values are passed to the script. If the script is written using an explicit command language, the variables are retrieved and set using the normal syntax for that particular language.

Figure 7.4 shows an example Perl script `PERL_SLEEP` with a numeric parameter `SECONDS` that sleeps for the indicated number of seconds. The Perl code shown is one line. Notice how it uses the parameter using variable syntax native to Perl: `$SECONDS`.

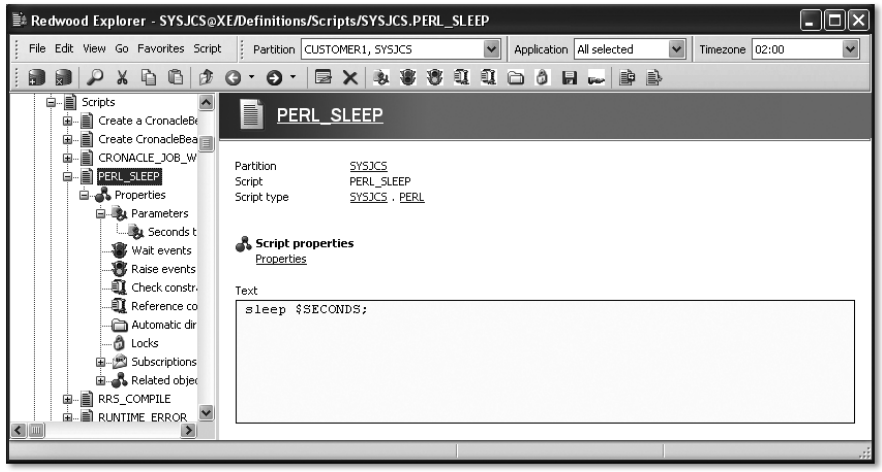


Figure 7.4 Using a Variable Value in Perl

PL/SQL Job Definitions

In SAP Central Job Scheduling by Redwood (full version), PL/SQL jobs are suited to advanced programmatic use of the Cronacle API, because the PL/SQL API is the core API in that release. Note that PL/SQL jobs are not allowed in the basic versions of SAP Central Job Scheduling by Redwood.

Figure 7.5 shows a sample PL/SQL written report job definition `RW_SHOW_INSTALLED_OBJECTS` that accesses the Cronacle API (view `jcs_component_version` and procedure `jcs_out.put_line` and `jcs_out.new_line`) to create a simple report.

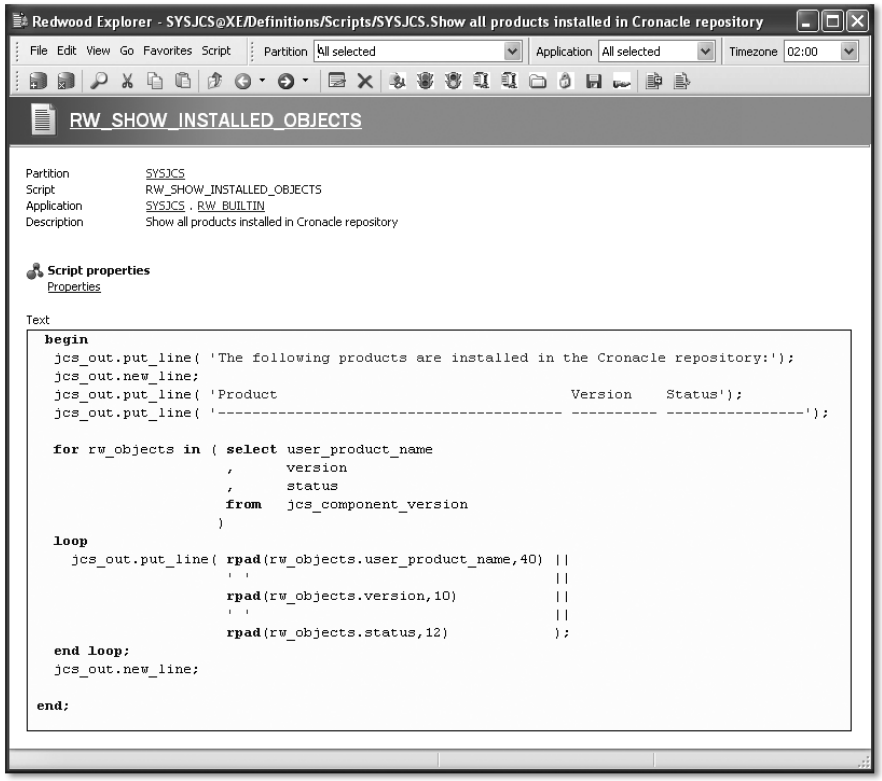


Figure 7.5 Example of a PL/SQL Report Job Definition

ABAP and BI Scripts

The Cronacle *for SAP solutions* interface creates scripts that call ABAP and BI jobs, as well as define new ABAP jobs. These jobs have an implementation that either refers to the `RSI` program or to the `SapR3` script type. Both have

the same effect: A connection is made to the Cronacle SAP RFC agent, which passes the request to the SAP instance. Both versions of SAP Central Job Scheduling by Redwood use these interfaces.

7.2.3 Job Chains

A job definition that consists of a job chain defines one or more sequential steps, each of them containing script calls executed in parallel. Job chains allow fast and easy development of complex job flows without losing the ability to incorporate site-specific specifications. Job chains are defined in the Redwood Explorer and do not require any programming.

A job chain with its sequence of steps is very similar to a CCMS job, but with parallelism, error, and event handling. SAP Central Job Scheduling by Redwood will convert multiple-step CCMS jobs to job chains when it imports these CCMS jobs. Figure 7.6 shows an example of how a CCMS job with two (ABAP program) steps is converted to a job chain with two steps.

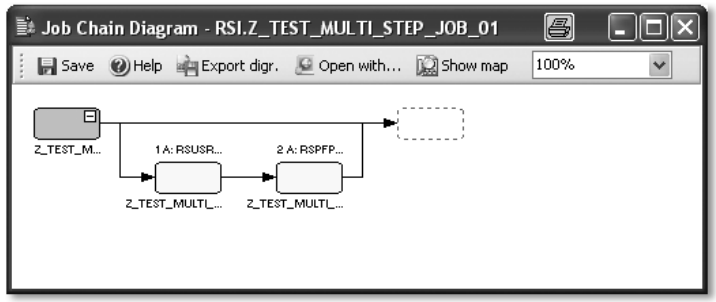


Figure 7.6 Example of an Imported Job Chain

It is a best practice to use job chains as much as possible where a stream of jobs is needed, as they are so well defined and can easily be run in parallel.

Steps

A step is a group of script calls that are started at the same time. A step has a status; the allowed statuses are shown in Table 7.1. A step may have one precondition that regulates whether the step is executed or skipped, and may have multiple postconditions that regulate what action is taken when the step is finished.

Step Status	Explanation
CHAINED	The job chain is requested and all corresponding jobs have been created but the step is not yet active.
WAITING	The step is started and one or more of its calls are currently active.
COMPLETED	All calls in the step were completed successfully.
ERROR	One or more calls in the step are not assigned status COMPLETED.
SKIPPED	The step was not executed because the pre-condition returned FALSE or NULL, or the START_AT_STEP parameter specified a later step, or a postcondition of an earlier step indicated that execution should continue at a later step.
DISABLED	The operator explicitly disabled this step. If the execution engine has not reached the step, it can be enabled again (in which case it will go to CHAINED.)

Table 7.1 Job Chain Step Statuses

A step has no output or log file. You can kill a step when it is in the WAITING status, which kills all running jobs within that step.

Start at Specified Step

When you submit and therefore request the execution of a job chain (at first-time execution or a restart), you can specify the name of the step where the process server should start by filling in the START_AT_STEP submit parameter. By default the job chain execution will start at the first step.

Preconditions

The execution of steps and script calls may be dependent upon specific circumstances, such as specific days of the week or the outcome of another job. In job chains, you define these conditions as pre-conditions to steps or script calls.

A precondition is a Boolean PL/SQL function stored in the Repository. If the outcome of the PL/SQL function evaluates to TRUE, the process server proceeds with the step or call. If the outcome evaluates to NULL or FALSE, process server skips the step or call and sets the job status to SKIPPED.

You also can skip a step or a call by manually changing the job status in the Job Monitor to DISABLED. These options should only be used when you exactly know what you're doing.

Postconditions

A postcondition specifies automatic or assisted error handling. It is normally defined at *step level*. It specifies what should happen given the final status of the step that just finished. You can define postconditions at two levels:

- Step level
- Job-chain level

A postcondition defined at *job-chain level* acts as default postcondition for all steps that do not define a postcondition handling the same status. A postcondition is triggered when one or more calls in a step or in the job chain get a specific job status.

The following job statuses can be used for postconditions (see Table 7.2).

Postcondition Status Handler	Explanation
`COMPLETED`	All calls in the step or in the job chain must be completed successfully before this postcondition is activated.
`KILLED, CANCELED, ERROR, UNKNOWN`	If any call in the step or in the job chain has the corresponding job status, this postcondition will be activated.
`OTHERWISE`	If there are no matches to the job status, this postcondition is activated.

Table 7.2 Postcondition Statuses

You can set a postcondition to cover a situation when the step is disabled by a pre-condition by using the `OTHERWISE` status. Upon reaching one of the job-states, you can do take of the following actions (see Table 7.3).

Action	Explanation
`GOTO`	Go to another step within the job chain.
`ERROR`	Raise an error and set a return code and error message.
`COMPLETED`	Change status to `COMPLETED`: ▶ On step level finish this step and set the step status to `COMPLETED` ▶ On job chain level finish the job chain and set the job chain status to `COMPLETED`
`RESTART`	Restart the job chain.

Table 7.3 Postcondition Actions

Action	Explanation
CONSOLE RESTART	Ask the operator what is to happen by sending a message to the Messages monitor.
CONTINUE	Ignore any failures and continue with the next step.
NULL	Ignore the default postcondition that is set for the corresponding postcondition on job chain level. This can be used at step level only.

Table 7.3 Postcondition Actions (cont.)

A step can be restarted automatically with the postcondition action RESTART or GOTO. When a step is restarted, the system will create new iterations of the steps that are to be run again. Steps or calls that were disabled in the last run (jobs with status DISABLED) will also be disabled in the new iteration. You can check what iteration a step is for by referring to the ITERATION column in the data-dictionary and the user interfaces.

Figure 7.7 shows a run of a more complicated real-life job chain that automates (a part of) the backup system in use at Redwood. During this run, several job steps failed, which is easy to see as their calls are colored red. Still within Redwood Explorer, we are able to zoom in on the job and see the actual error shown in Figure 7.8.

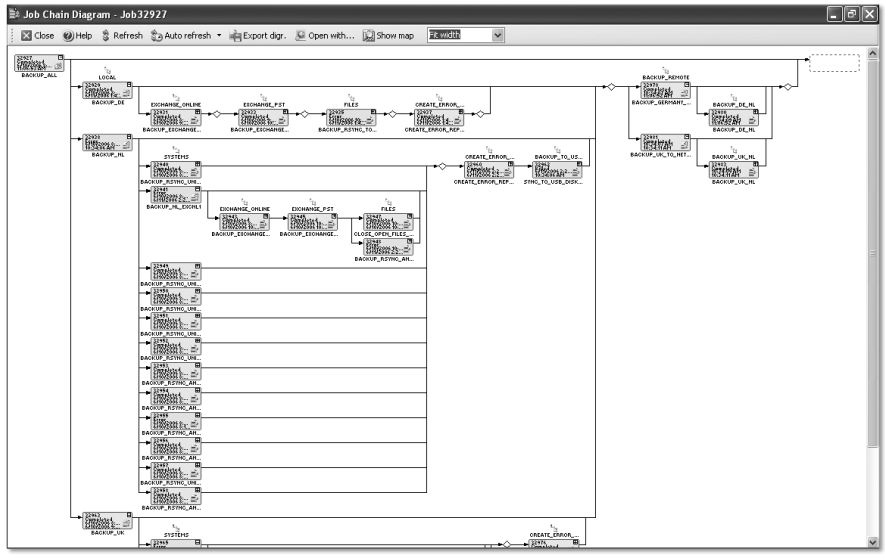


Figure 7.7 Example Run of a Non-Trivial Job Chain

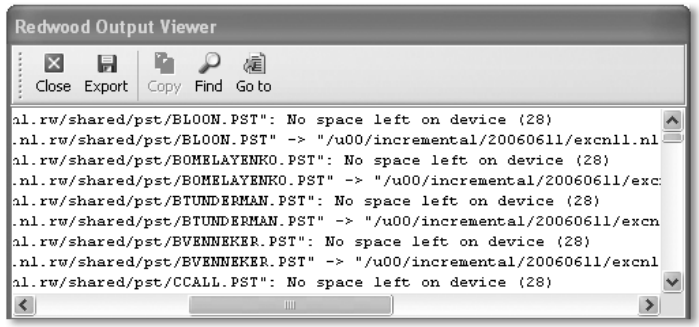


Figure 7.8 Output of Failed Job Shows Error Text

7.3 Job Allocation

Job allocation is an area that gets the full attention of SAP Central Job Scheduling by Redwood. It is not so important when only a single system is scheduled, because then all jobs need to run on the same node anyway. In a multiple node scheduling system, though, SAP Central Job Scheduling by Redwood needs to decide *which* jobs run *where* in *which order*. This is done in close cooperation with the job developer and the operator.

The job definitions specify the *priority*, an optional *resource,* and an optional target *queue.* The queues specify the minimal priority required for jobs to run. Schedulers (process servers) specify which queues they want to do work for, which *resources* (which will be explained in detail below) they accept, and how many parallel jobs per queue and minimal priorities per queue they require. Furthermore, schedulers need to explicitly support particular *script type*s. The appropriate job command language interpreter needs to be available on that node.

The result of these requirements is that only a subset of the schedulers is capable of running a particular job. The size of the queues limits the number of parallel running jobs. This means that at any given moment, the system always tries to fill all queues with as many running jobs as possible.

The complexity of the ordering is such that when many jobs are all queued and the system cannot process all of them, a choice must be made which job is started first. In SAP Central Job Scheduling by Redwood, the algorithm determines that the system will start the job with the highest priorities first. If it results in a tie (multiple jobs of the same priority in the queue in the `QUEUED` status) then it considers the parent's requested start time, and then

the job's requested start time. In other words, jobs that are part of a job chain that started earlier will start before new standalone jobs or new job chains. The underlying goal is to finish old work before starting new work.

7.3.1 Priorities

Priority can be set between **1** (lowest) and **100** (highest). The default job priority is **50**, with an increase of **2** for child jobs (per level). Job definitions can set the priority, and if they do then the priority of the job is fixed to that value. If the priority is not set at job definition level, it can be set during the job submit by the requestor.

Working with priorities is a good way of making sure that more important jobs will run first. Housekeeping jobs generally should have a lower priority than business processing jobs, so that they can be disabled easily when resources run out unexpectedly. Setting the priority right is especially important if you have situations where you stop all processing. After processing is started again, many jobs may have accumulated, either because of incoming business transactions or because time has passed. Consider which jobs you would like to run first: the important interface job or the lower-priority report that was submitted hours earlier. If you don't set the priority of your more important jobs, the system will start processing in chronological order.

7.3.2 Queues

A job is always assigned to exactly one queue. Unlimited numbers of jobs can be assigned to every queue; they are never "too full" to accept new work. You can create as many queues as needed.

As queues are securable items, you can grant users the capability to insert new jobs on a queue, to delete jobs besides their own ones, to update existing jobs, and to view all jobs on a queue-by-queue basis. You may want to create specific queues for specific user groups, e.g., a number of queues dedicated to a financial system that users outside the finance group have limited or no access to.

Queues have two major attributes:

- Queues can be *held*; in which case jobs can still be assigned to them but no job in the queue will start running. Already running jobs are not affected.
- Queues can specify a *size*. If it is set, then jobs will have to wait (job status remains `QUEUED`) until the number of concurrently running jobs drops

below the given limit. Job chains and steps are not counted towards this total, as they only claim virtual resources. The size can be set to one to get a simple exclusive execution mechanism, but locks are a better mechanism for that. The size is used to control the resource load on the system.

An important part of your design is deciding which queues to create and what sizes they should be defined for. When considering queues that run mainly or only CCMS jobs, a good rule of thumb is to allow one job in the queue per batch process in the SAP instance that it will run on.

Queues can also specify a maximum CPU load and/or maximum page rate. The queue will be closed automatically when the limit is reached and will be opened automatically when the CPU load drops below the limit.

Last, a minimum job priority can be set on a queue. Jobs lacking the required priority level have to wait (job status remains `QUEUED`) until the administrator lowers the required minimum priority or increases the job's priority. You can set a calendar time window to a job queue, limiting the opening time of the queue. The queue will be closed automatically when the calendar time window closes and will be opened automatically when the calendar time window opens.

7.3.3 Queues and Schedulers

A queue is served by one or more schedulers (process servers). If a queue is served by multiple process servers, jobs in that queue can be executed on one or more process servers.

You can define the same workload limits for the queue scheduler combination as you did for the job queue. As discussed earlier, workload limits will cause the queue to open and close automatically in order to manage the workload. The same thing will happen for the queue schedulers. Defining any workload limits for a queue scheduler will cause the queue scheduler to be enabled or disabled.

7.3.4 Resources

The resource object allows you to cluster schedulers that share certain characteristics. For instance, some of your OS schedulers may be able to access a tape robot and access data on those tapes. Or a particular job definition may only be able to run on one particular process server because it accesses something available only on that system.

In such conditions, you create resources and then specify those resources as requirements for job definitions. A job will then run only on those process servers that claim that they can provide the specified resources.

7.3.5 Script Types

Scripts that are stored in SAP Central Job Scheduling by Redwood in a particular command language need the appropriate interpreter or compiler before the script can run successfully. In such cases, the system also checks that the script can only run on process servers that support the required script type.

Please note that SAP Central Job Scheduling by Redwood (basic version) can only run scripts of the type `SAP`.

7.4 Job Management

Job management is concerned with the structures and capabilities that SAP Central Job Scheduling by Redwood provides to automate job management. Instead of manually observing conditions and saying "this job can run now," SAP Central Job Scheduling by Redwood provides a number of capabilities that can automate this.

The concepts that it provides to support are:

- Events
- Locks
- Time windows

7.4.1 Events

In most environments you must be able to react to *events* that happen completely out of the blue, or be able to instigate independent actions. Using events makes such actions explicit and enables a variable relationship between the actions that cause the event and the actions taken because of the event.

Note that it is unnecessary and may even be unadvisable to use events where they are not expressly needed. An example would be creating an application containing code that needs to do the following:

```
Read pricelist file from supplier system
Update SAP system with new price information
```

Since the actions that need to happen are quite obvious, it is just as easy to submit the appropriate CCMS job instead of raising a PRICELIST_UPDATE_REQUIRED or PRICELIST_FILE_ARRIVED event. Events are most useful where the code that raises the event is not aware of who will handle the event, so no direct relationship should exist between one or more jobs.

Raising an Event

In SAP Central Job Scheduling by Redwood, there are six typical sources of events:

- **From an external source (full version only)**
  A command line utility rsno allows generation of events from almost anywhere.
- **From an SAP event**
  The SAP interface can translate internal SAP events into SAP Central Job Scheduling by Redwood events. This functionality requires either XBP 3.0 or the use of rsno.
- **From a file (full version only)**
  The arrival of or the change of a file on a system where a process server is running.
- **From a status change of a job**
  A job definition can define raise events that specify which event should be raised when a job instance of that job definition reaches a particular status. When that status is not reached, the event is not raised. For instance, a job definition can raise the WARN_OPERATOR event when an important job fails.
- **From a call of the jcs.raise API procedure (full version only)**
  The full version allows the user to define PL/SQL user exits in the form of user-defined triggers, PL/SQL job definitions and other PL/SQL code. This code can access the Cronacle API jcs.raise directly.
- **From within the (graphical) user interface**
  The user can explicitly raise and clear events using the Redwood Explorer and Redwood Process Manager for Web.

  Figure 7.9 shows how this is typically used. The job chain HANDLE_PRICELIST_CHANGE is depicted, consisting of a READ_PRICES step followed by a PROCESS_PRICES step. Both steps will raise an event (the "green traffic

light" icons) when the job results in an error. The job chain itself will automatically be submitted when an event PRICELIST_ARRIVED is raised (the "red traffic light" icon).

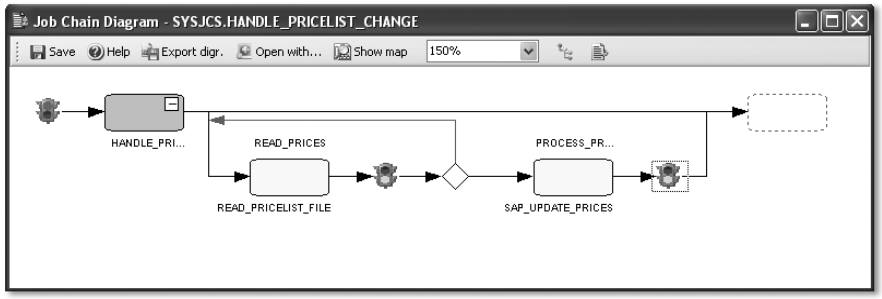


Figure 7.9 A Job Chain Waiting for and Raising Events

7.4.2 Locks

To prevent jobs running concurrently—for example, when two or more jobs need access to the same system resource, such as a tape unit—you can define *job locks*. A job requests a job lock before the job is able to start; that is, before the job status can change from QUEUED to RUNNING.

The job status becomes LOCKWAIT when a lock is already obtained by another job that has the status RUNNING, WAITING, or CONSOLE, and the requesting job must wait. The status of the requesting job changes to RUNNING when the other job releases the lock and the requesting job gets the lock. Multiple locks can be defined for a job definition and can be used to help define which jobs can and cannot run together.

Example

You have three jobs to be run. Job 1 should run when 10 orders are entered and must run without Job 2 or Job 3 running as well. Job 2 should run when a file arrives and may not run simultaneously with Job 3. Job 3 should run on demand from end-users.

Although simple "exclusive" conditions can be emulated using job chains or queues with a restricted size, this becomes problematic when the run frequencies of the jobs are not similar, as in the example just given. The example can be modeled using two locks: LA and LB, Job Definition 1 demands lock LA only, and Job Definitions 2 and 3 specify LA and LB.

7.4.3 Time Windows

A *time window* defines *when* something is allowed. Closely related to time windows is the time zone where the job is being scheduled in. Time zones are described in Section 7.5.2.

A time window is defined by one or more time intervals. Each time interval defines a period of time by fixed dates and times or rules such as "the first work-day of the month," or a combination of both (dates and times and rules). A set of time intervals need not be continuous.

The system supports nested time windows. A time window can be enabled within another time window, or it can be disabled during a specific time window.

Example

A time window `NATIONAL_HOLIDAYS` is defined to include days such as New Years Day, Easter Monday, Christmas Day, etc. (predefined definitions for this come with the system). Once this single time window is defined, it can be used in other time windows, e.g., `FACTORY_HOURS`, `OFFICE_HOURS`, and `STORE_HOURS` can all specify that they are not open during `NATIONAL_HOLIDAYS`.

Time windows are managed centrally. To use time windows, you must have the appropriate privileges. The options to define time windows, during which job execution is allowed, are quite extensive. Consequently, the definition of time windows on multiple levels can lead to very complex and sometimes conflicting situations. The system detects these conflicts.

If, for example, a job with the time window `WORKING_DAYS` is started in a queue with the time window `WEEKEND`, the job will never actually be started. The process server detects such a situation and sends a message to the job monitor by setting the job status to `NEVER` and the start date to December 31, 2999.

A time window can be specified for the objects shown in Table 7.4.

Object Type	Use of the Time Window
Job	Job can only be executed within this time window.
Job definition	A job based on the script is executed within the time window. This setting can be made mandatory, meaning it cannot be overruled when requesting the execution of the job definition.

Table 7.4 Possible Uses for Time Windows

Object Type	Use of the Time Window
Queue	The queue is open only within the time window. If it is outside the period specifying the time window, the queue is automatically closed. The time window set for a queue influences the start date and time or time window of a job.
Queue scheduler	A process server is enabled for the queue only within the time window. Outside the period specifying the time window, the process server is automatically disabled for the queue. You can enable it for other queues, however.
Submit frame	A repetitive job cannot be started unless the next cycle lies within the time window. If the next cycle lies outside the time window, the job is automatically postponed to the next moment the time window is open.

Table 7.4 Possible Uses for Time Windows (cont.)

7.4.4 Submit Frames

A *submit frame* is a cyclic calendar which forces a job to be executed repetitively. When script execution is requested and a submit frame is used, the corresponding job will be restarted automatically by a process server according to the frequency defined in the submit frame. You must have the appropriate privileges for using a submit frame.

Figure 7.10 shows the built-in submit frame `RW_EVERY_30_MINUTES`, which can be used to submit a job every 30 minutes.

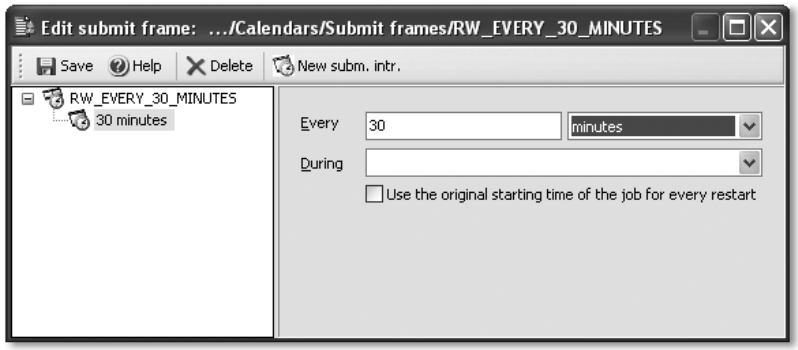


Figure 7.10 Example of a Submit Frame

You also can request multiple executions of a script by specifying a recurrence. A *job recurrence* is specified for one sequence of repeating jobs when

requesting the execution of a script. A submit frame, on the other hand, is an object that can be used for multiple execution requests. You should use recurrent jobs unless you need a separate calendar object or when you require the job to run more often than once a day.

Figure 7.11 shows a recurrence where the job is scheduled to run every Friday from April 16, 2006 onwards.

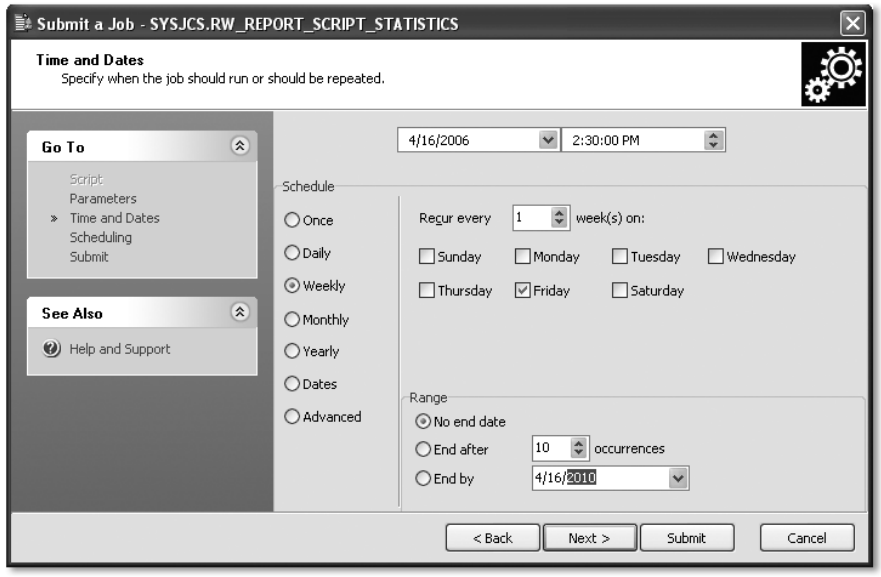


Figure 7.11 Example of a Recurrence

7.5 Job Submission and Monitoring

All dependencies between jobs and how and when they are submitted are explained in this section.

7.5.1 User Parameters

During the job submit, the user enters actual values for all parameters. The constraints are checked to verify that the user enters valid values, as well as used to provide drop-down lists (or value-selection dialogs). An example of how complex user parameters can get, is shown in Figure 7.12.

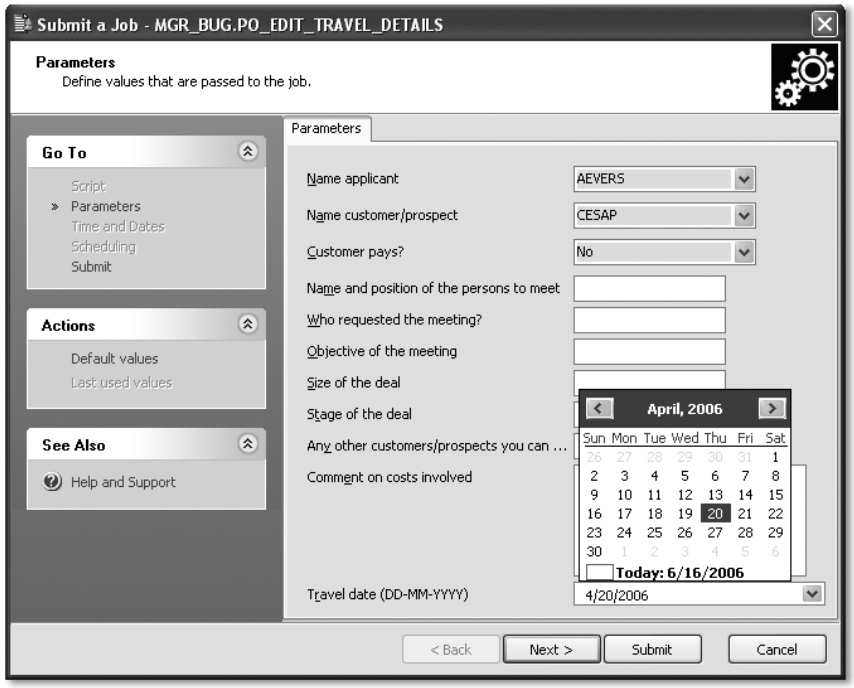


Figure 7.12 Submit Dialog with User Parameters

7.5.2 Schedule Parameters

Every script has scheduling parameters, and the default values can be changed before the execution request is passed on to the process server. The values are optional because they all have a default value. The scheduling parameters shown in Table 7.5 can be set.

Scheduling Parameter	Technical Name (API)	Explanation
Queue	QUEUE_NAME	A queue can be set to ensure that jobs are processed according to priority on the server computer. It is not editable if the script has a preset queue.
Start job at	STARTTIME	The date and time when the job must be executed. You cannot indicate a point in time (date and time) that has already passed. The earliest start time is always the current date and time.

Table 7.5 Scheduling Parameters

Scheduling Parameter	Technical Name (API)	Explanation
Pattern for recurrent jobs	RECURRENCE_DATA	Use a recursive pattern when you want to request the job to be run several times. You cannot use a submit frame and a recurrence pattern at the same time.
Priority	PRIORITY	Execution priority may be set for each job. Priority cannot be increased if it is predefined.
Status when job is submitted	STATUS_ON_SUBMIT	Use this parameter to request jobs to be executed on status HOLD instead of the default status, SCHEDULED.
Start at specified Step	START_AT_STEP	This scheduling parameter is available for job chains only. When you request the execution of a job chain (first time execution or a re-request), you can specify the step name where a process server should start. Job-chain execution will start at the first step if a step name is not specified.
Time window	TIME_WINDOW_NAME	The start time also can be determined by setting a time window to the job. Time windows always overrule any explicitly set start date and start time. It cannot be changed if it is preset.
Submit frame	SUBMIT_FRAME_NAME	Activate repetitive job execution by setting a submit frame. A submit frame defines a cycle of repetition. The Job Monitor always shows the first scheduled job of a repetition. Use the **Calendar** view in the Job monitor to get a complete forecast of job repetition. You cannot use a submit frame and a recurrence pattern at the same time.

Table 7.5 Scheduling Parameters (cont.)

Scheduling Parameter	Technical Name (API)	Explanation
Repeat when job repeats	`REPEAT_WITH_PARENT`	When output publishing is requested for a job, the process server executes a destination job. Destination jobs have a special scheduling parameter called `repeat when job repeats`, which specifies whether to re-execute the output destination when the parent job (the job that produced the output) is restarted.

Table 7.5 Scheduling Parameters (cont.)

Sources of Scheduling Parameters for a Job

The scheduling parameters for a job (such as queue, time window, submit frame, priority) are derived from many sources. For queues, the actual queue used can be identified by the first result that gives a queue name:

1. An argument set in the `jcs.run`, `jcs.submit`, `jcs.reschedule` or `jcs.parameter` call (full version only).
2. The queue as defined at script level.
3. The queue of the parent job.
4. The queue as set by `jcs.setqueue`.
5. The pre-defined queue `SYSTEM`.

Note

If either Result 1 or Result 2 from this list results in a queue, and so does Result 3, and they are not equal, then an exception will be thrown at the final submit, run or reschedule call.

Time zones

The SAP Central Job Scheduling by Redwood Release 7.0 supports the definition of multiple user-defined time zones. This is done so that the system administrator can limit the number of time zones used within the system. The data is stored using official time zone names such as **UTC +01:00**, **US/Eastern** or **Europe/Berlin**. It is displayed with more user-friendly names such as **New York**, **Walldorf,** or **Houten**.

It is quite important to use actual time zones correctly. Otherwise, unexpected time shifts may happen when a region goes into or out of daylight savings.

Example

If you want to schedule a job that is allowed to start at close of business in New York (6 PM local time), you should use the **US/Eastern (New York)** time zone and no other. If you schedule it to run at 6 PM in time zone **–05:00** in winter, it will run at 7 PM local time in New York once Daylight Savings Time goes into effect (when local time is **–4:00**).

7.5.3 Forecasting

When recurrences are used, the future job executions are already present and the job monitor calendar view will show these automatically. When you use submit frames only, the "next" execution will be seen in the job monitor.

If you want to predict when jobs using submit frames will be executed, it is possible to make a *forecast*. This functionality is provided by the Forecast module, which is not installed by default. You have to install it by running the module installer script `RW_FORECAST_INSTALL` (Redwood Explorer: **Definitions • Applications • FORECAST • Scripts • RW_FORECAST_INSTALL**).

Once you have installed the forecast module, you can either perform a forecast manually by running the process server system script `RW_FORECAST_JOBS` (Redwood Explorer: **Definitions • Applications • FORECAST • Scripts • RW_FORECAST_JOBS**) or let the process server generate it continuously by running the Redwood system script `RW_SET_FORECAST_JOB_OPTIONS` (Redwood Explorer: **Definitions • Applications • FORECAST • RW_FORECAST_SET_JOB_OPTIONS.**)

The jobs that are forecast will appear in the **Calendar** view of the Job Monitor based on time windows, submit frames, and average run times. Continuous forecasting does place an extra load on the system and should be carefully managed on systems that are nearing maximum throughput.

Figure 7.13 shows a screenshot of the job monitor in **Calendar** view (Jobs by week) mode. The semi-transparent jobs are forecasted jobs; when you select them, the status bar will show which job the forecast is for, and its name. This job will also be shown in the job list pane underneath the calendar.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X