# RPG Maintenance Utility (GUI)

## Introduction

The RPG maintenance utility (GUI version), called “the Utility” in this document, has the purpose of maintaining configuration data for the main RPG application. There are three sets of configuration data to be maintained:

* Job schedules
* Database server access data
* General configuration settings

The Utility facilitates adding, deleting or modifying configurations.

## GUI Window Design Requirements

1. The Utility should offer three list views for the various kinds of entities to be managed. The window should have buttons or tabs to switch between the three views:
   * Jobs
   * Servers
   * Configuration Items
2. All views should have the following action buttons:
   * Add … allows adding a new item in a pop-up window.
   * Change … allows changing a (selected) item in a pop-up window pre-filled with the current values and an ensuing syntax check.
   * Delete … allows deleting a (selected) item after a confirmation dialogue.
3. The Jobs list should have the following columns:
   * Name … containing the job name
   * Last Run … containing the timestamp of the last execution
   * Next Run … containing the timestamp when the job runs next
   * Frequency … A textual representation of the job execution schedule
4. The Servers list should have the following columns:

* Name … Name of the server
* Address … The IP address or DNS name of the server
* Port … The port of the server
* Type … Denoting the server type
* User … The username to be used when logging on to the server

(Note: the password should not be shown in this view.)

1. The configuration item list should have two columns:
   * Parameter … Name of the configuration parameter
   * Value … The value (or “(encrypted)” when the value is encrypted).
2. All windows should have a “Cancel” button or an “OK” button if an action is about to be performed that changes any data. “Cancel” should allow top back out without any impact, while “OK” performs the action.   
   Exception: For purely informational/message windows where no action is pending, an “OK” button is sufficient and no “Cancel” button needs to be added.
3. Job data entered by the user should be syntax checked as follows:
   1. Name: Converted to uppercase, 4-12 characters long, regular expression format options: /[A-Z]{4,12}/ or /[A-Z]{2,8}[0-9]{2}[A-Z]?/ . When adding to the list of existing jobs, check for duplicates.
   2. Freqency: A three letter abbreviation for a weekday (MON, TUE, WED, THU, FRI, SAT, SUN) or an integer in the range 1 to 28. Allow for case-insensitive entry and convert to uppercase before parsing.
4. Server data entered by the user should be syntax checked as follows:
   1. Name: Converted to uppercase, 4-16 characters long, alphanumeric. When adding, check for duplicates.
   2. Address: A syntactically valid IP address (use ipaddress module to parse) or host name (do not try to resolve the hostname when entered; use regular expression parsing to check)
   3. Port: A valid port number, integer between 1 and 32767
   4. Type: Always “oracle”, “mssql” or “api”. The window should provide radio buttons to select one of these options.
   5. User: A string containing the username. Should not be blank, but no further syntax checking or case conversion should be applied.
   6. Password: Should not be blank, but no further syntax checking or case conversion should be applied. Do not display the password when entered.
5. Configuration items entered by the user should be syntax checked as follows:
   1. Name: Converted to uppercase, 4-12 characters long
   2. Value: Should not be blank, but no further syntax checking or case conversion should be applied.
6. Coding requirements:
   1. Use the wxPython package for the GUI logic.
   2. Use the black formatter to format the code.
   3. Use pylint with the provided .pylintrc file to lint the code.
      1. The code should be free of errors and warnings.
      2. If necessary, add exclusion clauses to individual lines.
   4. When in doubt, follow the PEP 8 coding style.
   5. Add comments describing the purpose (result or side effect) of code sections.

## RPG Configuration class

The RPG configuration class provides utility functions to handle RPG configuration items. Its use is always preferred over manual access to the INI file. For reference purposes, the configuration file contents are described below.

When instantiated, an RPGConfig object contains the configuration to be used. The following methods are provided:

|  |  |
| --- | --- |
| Method | Description |
| delete\_job | Delete a job from the configuration |
| delete\_server | Delete a server from the configuration |
| get\_job | Get the job details for a given job ID (is\_due, day, last\_run, next\_run) |
| get\_job\_day\_text | Get the textual (human readable) version of the run schedule for a given job ID |
| get\_param | Get a parameter value |
| get\_server | Get the server details for a given server ID (hostname, port, username, password, type) |
| has\_param | Determine if a parameter exists |
| job\_exists | Determine if a job exists |
| job\_is\_due | Determine if a job is due to run |
| jobs | Return the ids of all defined jobs |
| parameters | Return all parameters as a dictionary |
| reset\_job | Reset the last\_run attribute of a job so that it is due to run at the next scheduled time |
| run\_job | Update the last\_run attribute of a job to the current time |
| save | Write the current configuration to the file |
| server\_exists | Determine if a server exists |
| servers | Return the ids of all defined servers |
| set\_job | Set the execution schedule for a given job ID |
| set\_param | Set a parameter value |
| set\_server | Set the server details for a given server ID |

### Configuration data storage

The configuration data is stored in an INI file called rpg\_ods.ini, with each job schedule and server configuration item in an individual section. General configuration settings are maintained in the CONFIG section.

The type of configuration data is identified by the section name. The section names in the configuration data file use the following scheme:

|  |  |
| --- | --- |
| CONFIG | General configuration settings |
| JOB:xxxxx | Job settings (xxxxx contains the job name) |
| SERVER:yyyyy | Server access data (yyyyy contains the server name) |

Job sections have the following items:

**cron** – this contains a five-element entry cron expression[[1]](#footnote-1). RPG uses only the day of month (third)and day of week (fifth) part of the cron expression for scheduling purposes. The first, second and forth element is always a single asterisk. This element is mandatory.

**last\_run** – The time when the report had been last run. This element is optional and may be missing. If it is not present, this means that the report had never been run. If it is present, the time stamp of the last run is stored in the format "%Y-%m-%d %H:%M:%S" and in the local time zone (Mountain Time).

Database server access sections have the following items:

**address** – The IP address or DNS resolvable name of the server (mandatory)

**port** – The port number to be used (integer, mandatory)

**user** – The username to use when connecting to the server (mandatory)

**password** – The encrypted version of the password to use when connecting to the server. The configuration object manages local decryption and encryption without the need to perform it.

**type** – Describes the server type: ORACLE, MSSQL or API

1. For the definition and interpretation of cron expressions, see [*https://www.baeldung.com/cron-expressions*](https://www.baeldung.com/cron-expressions) and [*https://en.wikipedia.org/wiki/Cron#Cron\_expression*](https://en.wikipedia.org/wiki/Cron#Cron_expression)*.* [↑](#footnote-ref-1)