CODE MAGUS

cvsci: CVS Check-in Validation Configuration Reference

CML00012-01

Business Partner IBM™

August 16, 2016

Contents

1 Introduction.

1.1 Overview.

The cvsci program runs at `CVS` check in time. It will validate the parameters passed to it by `CVS` and return success or failure based on a given configuration. The configuration defines the rules to which the path and name of a file being checked in must adhere to.

2 Program execution.

2.1 Parameters.

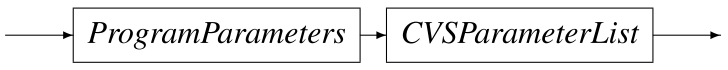
2.1.1 Overview.

At execution time `CVS` passes the `path` and `file names` of files to be checked in to the program as parameters. Although these parameters are defined by `CVS` the program does allow other parameters first. When implementing this program and specifying it in the `CVS` file `commitinfo` these parameters can be specified; `CVS` will append the `path` and `file names` after these.

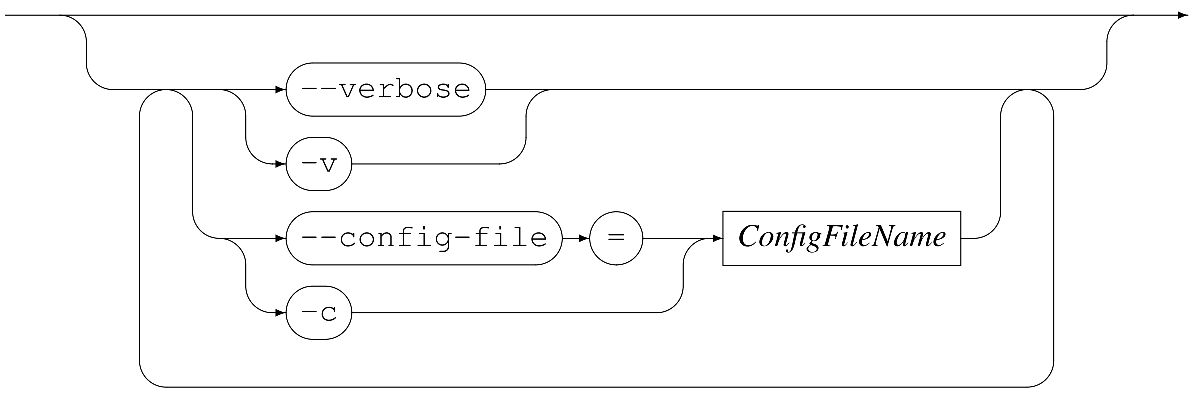
2.1.2 Parameter syntax.

The parameters are set out as follows:

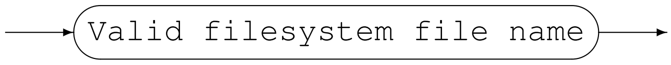
Parameters



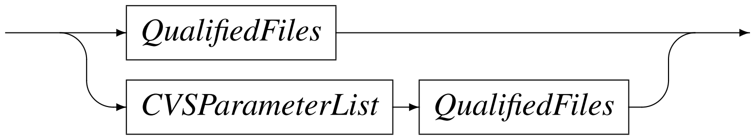
ProgramParameters



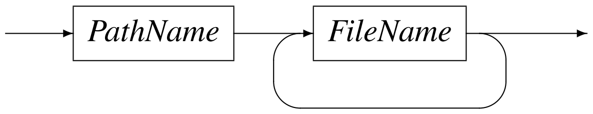
ConfigFileName



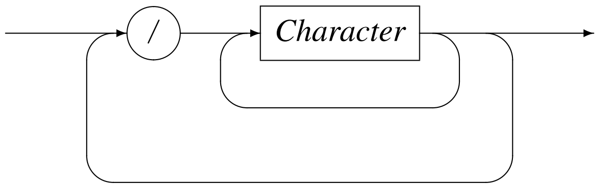
CVSParameterList



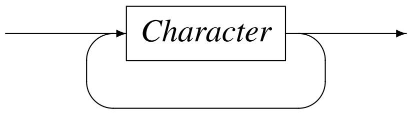
QualifiedFiles



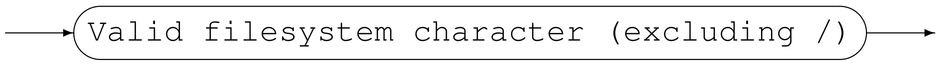
PathName



FileName



Character



2.1.3 Examples

- Single directory and file

  ```
  ./cvsci /Pathnode1/Pathnode2 file11
  ```

- Single directory and file, plus config file definition

  ```
  ./cvsci --config-file=./cvsci.cfg /Pathnode1/Pathnode2 file11
  ```

- Single directory and multiple files

  ```
  ./cvsci /Pathnode1/Pathnode2 file11 file12 file13
  ```

- Multiple directories and files This command is split into two lines for readability only.

```
./cvsci /Pathnode1/Pathnode2 file11 file12
        /Pathnode3/PN4/PN5 f21 f22 f23 f24
```

3 Configuration.

The execution of the program is controlled by a configuration file and various environment variables.

3.1 Environment variables.

The environment variables that affect program execution are:

- `CVSCI_CONFIG` This defines the configuration file that the program will use
- `CVSCI_VERBOSE` If set to Y will cause the program to emit detailed information about its progress.
- `CVSCI_DEBUG` If set to Y will cause the program to emit very detailed information about its progress. This option prints developer information and should not need to be used for normal processing in order to understand the execution flow or diagnose a problem.

3.2 Configuration file.

The configuration file holds all the information needed by the program to validate each node of the path and then finally the file name as presented by `CVS` .

3.2.1 NODE entry

There has to be a node called `Root`. As the program validates a path name it always starts with the node called `Root`. Each node defined in the configuration file is made up of a number of rules. These rules are each associated with a regular expression that is used to match the node name being validated and therefore determine which rule is to be used.

The types of node defined in the configuration file are:

- Path nodes. These nodes always refer to a next validation node. This means that as the program validates each node the rule used defines which set of rules (or validation node) to use for the next node of the given path name.
- File node. These nodes are defined by the fact that they never point to another node. I.e. they are terminating nodes for the rule tree. Each file name is matched to check that it conforms to one of the regular expressions defined by the rules within that node.
- Special nodes. These nodes do not have specific rules but keywords that determine the processing allowed. They are:
  - allownone - No files or directories are allowed to be checked in under this node.
  - allowall - Any name format can be used for files or sub-directories under this node.
  - empty - The node allows no specifically named files or sub directories under it except those matching the rules defined in the common node.

Nodes need to be defined in such a way that there are exactly the same amount of nodes in the path name being validated as in the validation nodes defined in the configuration file. The last node of the path must reference a node that will validate the file name. I.e. it is considered an error if the number of nodes made up by the path and file name from `CVS` does not exactly match the number of defined validation nodes in the configuration file.

3.2.2 NOTIFYFILE entry

This entry allows the system administrator to configure contact information, so that when a user receives an error message due to some problem they also receive the contents of the named notify file. So if a user cannot solve their problem or they have experienced a system type failure, they can use this information to contact or notify the correct person. Each line of this file is echoed back to the `CVS` user except those starting with `--`; such lines are comment lines in the notify file, usually there for the convenience of the administrator setting up the system.

3.2.3 COMMON NODE entry

The common node entry allows for the definition of terminal node rules that will apply to all file names from CVS. In other words once validation reaches the file name, the file name is validated against the current validation node's rules and if there is no match also against the rules for the common node. Only if both fail is the file name rejected as not matching the supplied rules. Consider the following example.

Register the common node name:

```
common node common_files;
```

This means that the rules for the node '`common_files`' will also be checked when a file name is validated.

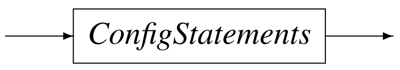
Define the node as follows:

```
   NODE common_files
      description
      "The common node rules are used for validating every node"
      "after the node's own rules are validated for a file name."
      "This can be helpful when there is a file (or pattern name)"
      "that occurs in every directory of the tee structure."
      "An example is 'readme.txt'"
      RULE README MATCH /^readme\.txt$/
   ;
```

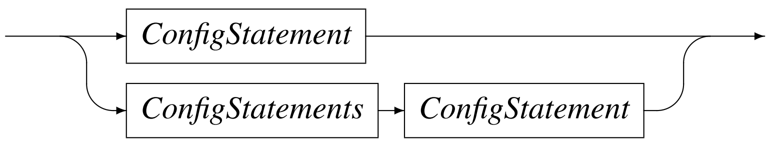
This node does not have to be pointed to by any other node in the tree. In other words it is usually a stand alone node that is only accessible through the 'COMMON NODE' configuration statement.

3.2.4 Syntax.

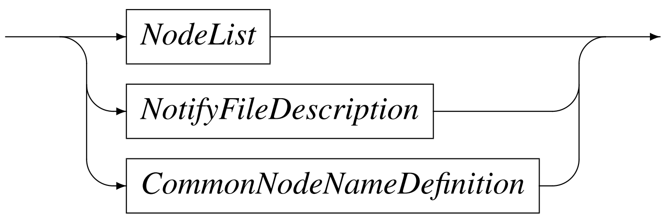
ConfigFile



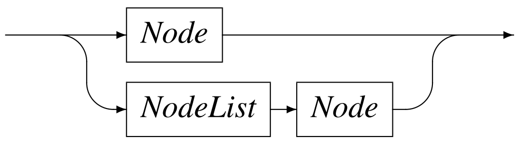
ConfigStatements



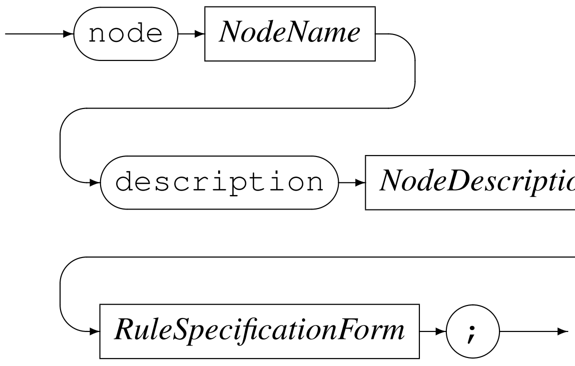
ConfigStatement



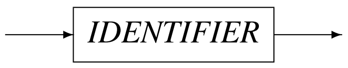
NodeList



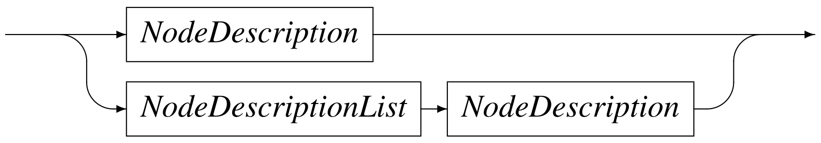
Node



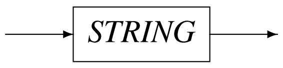
NodeName



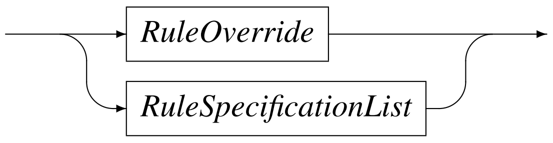
NodeDescriptionList



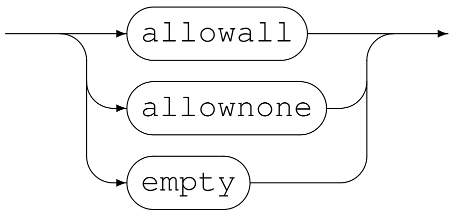
NodeDescription



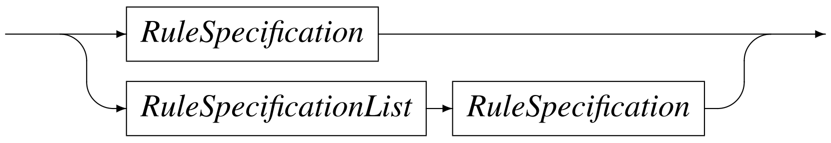
RuleSpecificationForm



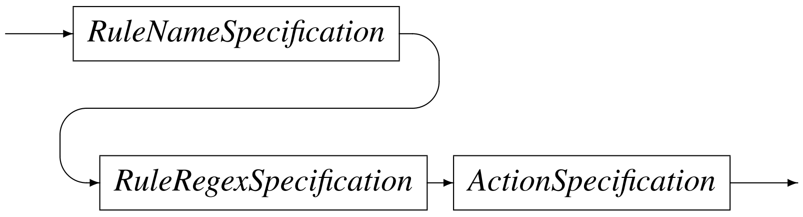
RuleOverride



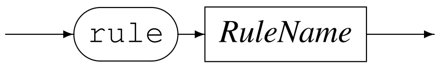
RuleSpecificationList



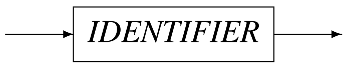
RuleSpecification



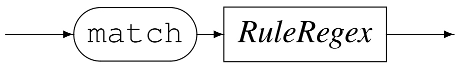
RuleNameSpecification



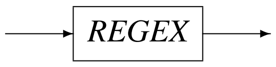
RuleName



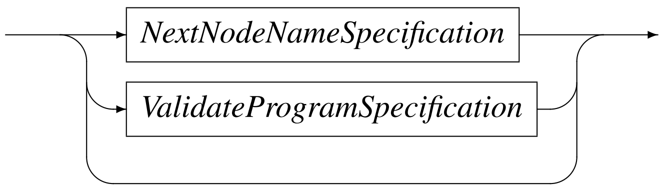
RuleRegexSpecification



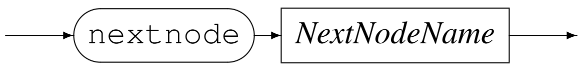
RuleRegex



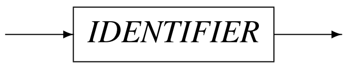
ActionSpecification



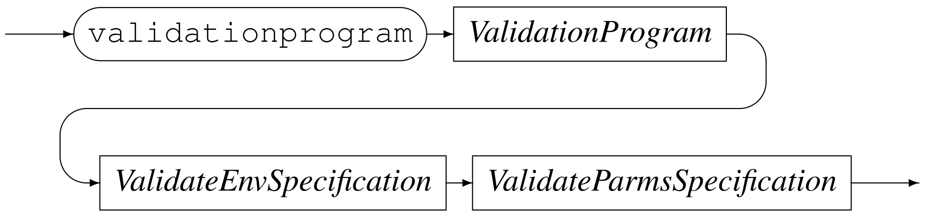
NextNodeNameSpecification



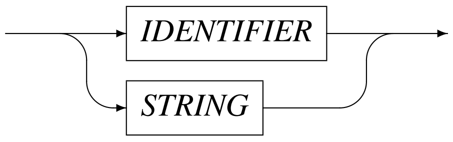
NextNodeName



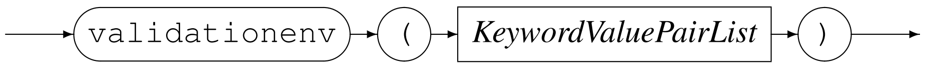
ValidateProgramSpecification



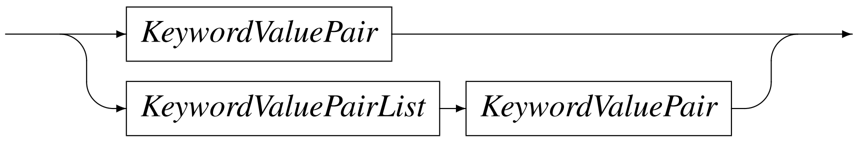
ValidationProgram



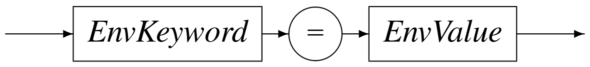
ValidateEnvSpecification



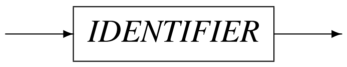
KeywordValuePairList



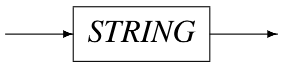
KeywordValuePair



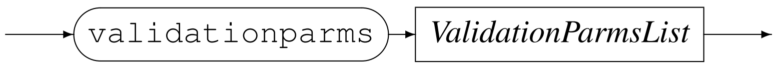
EnvKeyword



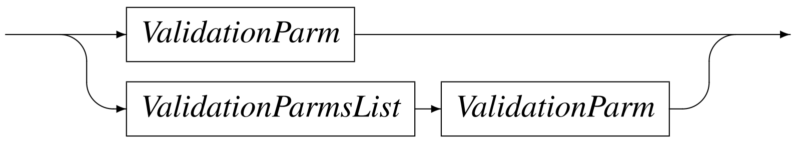
EnvValue



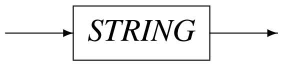
ValidateParmsSpecification



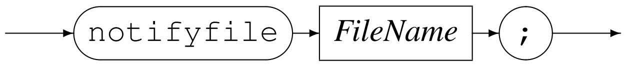
ValidationParmsList



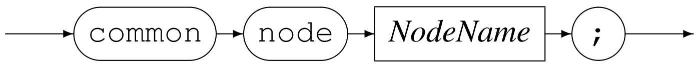
ValidationParm



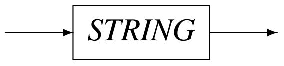
NotifyFileDescription



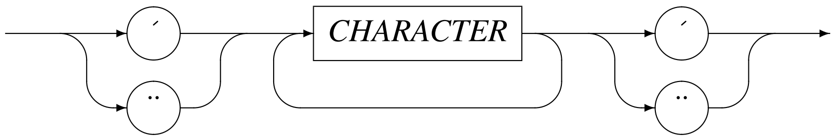
CommonNodeNameDefinition



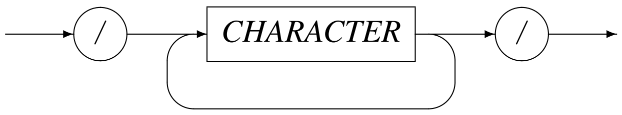
FileName



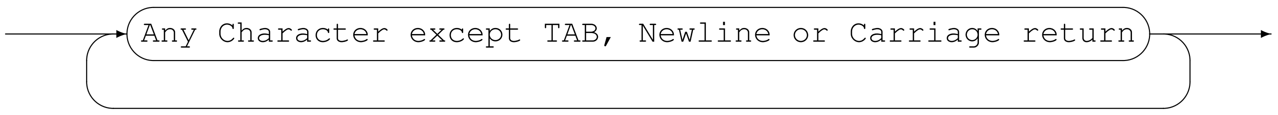
STRING



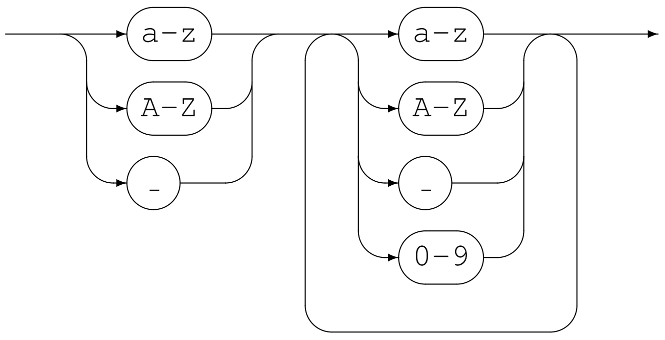
REGEX



CHARACTER



IDENTIFIER



4 Developer Testing.

Within in the CVS module cvsci there is a directory called testing. The shell script test_cvsci.sh can be used to test the execution of the program. There is a section for negative tests (e.g. invalid configuration file syntax etc.) and positive tests (e.g. testing multiple paths and file names). This script and the corresponding configuration file test_cvsci.cfg should be maintained if and when necessary.

A Configuration examples

1. **Notify file specification**

   ```
   notifyfile /home/cvs/CVSROOT/cvs_notify.txt
   ```

2. **Root rule - Required node validation** This example defines the first validation node. The node must be called Root (and is case sensitive). In this example /home must start the name of every directory passed to this program. The next directory name is defined by the rule CVSNODE.

   ```
   NODE Root
      DESCRIPTION
      "The Root Node is the first node in"
      "the tree and must be specified exactly."
      RULE rule01 MATCH /^home$/ NEXTNODE CVSNODE
   ;
   ```

3. **Directory validation node - next node only.** This validation node defines that the directory name must be cvs and that it must be followed by another directory defined by the rules in the node MODULEA.

   ```
   NODE CVSNODE
      DESCRIPTION
      "This is the main CVS node. Only specified"
      "sub-directories are allowed under it."
      RULE rule02 MATCH /^cvs$/ NEXTNODE MODULEA
   ;
   ```

4. **Directory validation node - file validation only.** This validation node defines a directory that must have the name spreadsheets and only files of type .xls may be checked into it. It may not have any sub-directories.

   ```
   NODE spreadsheets
      DESCRIPTION
      "This is the spreadsheet directory"
      "File names must start with A-Z or a-z"
      "followed by any number of A-Z, a-z, 0-9"
      "or _ (underscore) and must end with .xls."
      RULE rule03 MATCH /^[A-Za-z][A-Za-z0-9_]*\.xls$/
   ;
   ```

5. **Directory validation node - file or directory validation.** This validation node defines a directory that must have the name moduleA and allows the next directory in the path to be defined by the

rules of node `spreadsheets` or a file in this directory to be named as per the file rule (namely `dir.txt`).

```
NODE MODULEA
   DESCRIPTION
   "This is the moduleA directory"
   "It only allows the directory defined for"
   "spreadsheets and a file named dir.txt."
   RULE rule04 MATCH /^moduleA$/ nextnode spreadsheets
   RULE rule05 MATCH /^dir\.txt$/
;
```

6. **Directory validation node - Override values.** These validation nodes define a directory that:

   - Allows no files or directories beneath it. Often this type of validation node is used as a place holder for eventual (correct) rules. This also overrides the common node rules.

     ```
     NODE NODE1
        DESCRIPTION
        "This directory allows no file or sub-directory name under it."
        allownone
     ;
     ```

   - Allows any files or sub-directories beneath it.

     ```
     NODE NODE2
        DESCRIPTION
        "This directory allows any file or sub-directory name under it."
        allowall
     ;
     ```

   - is empty. Effectively this allows no files or sub-directories, except those that are defined in the common node.

     ```
     NODE NODE3
        DESCRIPTION
        "This directory allows files and directories that match the"
        "rules defined in the common node (if it is defined)."
        empty
     ;
     ```

7. **Specifying the common node.** The following defines which node definition will act as the common node:

   ```
   common node rules_for_all
   ;
   ```

   The following validation node defines the rules for the common node; in this case any file called “`readme.txt`”. For a detailed explanation see section 3.2.3 on page 5.

   ```
   NODE rules_for_all
      DESCRIPTION
      "This is the common node rules specification."
      RULE rule05 MATCH /^readme\.txt$/
   ;
   ```