

# SPCL: Structured Policy Command Language Reference Manual

[Michael Locasto](#) [Matthew Burnside](#) [Chun Li](#) [Aron Wahl](#)

Department of Computer Science  
Fu Foundation of Engineering & Applied Science  
Columbia University  
{locasto,mb@cs.columbia.edu}  
{alfredli1,aronwahl@hotmail.com}

---

## 1. Introduction

This page is a description of the syntax of Structured Policy Command Language (SPCL). Before proceeding to the details of this language, it is highly recommended to familiarize yourself with the overview and the goal of SPCL.

The overall structure of SPCL is as follow:

- Each SPCL file is associated with a distinct [policy](#) under a finite administrative domain, known as [zone](#).
- Each [policy](#) is composed of [objects](#), [principals](#), and [groups](#).
- [Objects](#) could either be some representation of the environment or some resources of the system to be accessed by users.
- [Principals](#) are the entities or users of the system. Associated with each [principal](#), among with others, are a set of rules describing the parts of the resources that could be accessed or the parts that are restricted. There is a [default](#) principal describing the privileges of any unidentifiable [principals](#).
- [Groups](#) exist solely for the convenience of the administrator for organizing [principals](#) with the same privileges under the same [zone](#).

Again, the above discussion is just a brief abstraction of SPCL. For further details, please refer to the corresponding sections below.

## 2 Lexical Convention

There are five kinds of tokens: keywords, identifiers, numbers, strings and operators. In general, as in most languages, white spaces are ignored except as they serve to separate tokens. When parsing tokens, the longest matched string of characters is used, unless otherwise specified.

### 2.1 Comments

A comment is a sequence of characters served as notes for human beings, and is ignored by the compiler. Two forms of comment are available.

1. Line comment, which starts with `//` and end with the first occurrence of a line terminator.
2. Block comment, which starts with `/*` and end with the first occurrence of `*/`.

## 2.2 Identifiers (Names)

An identifier is a sequence of letters, underscores and digits, with the first character being a letter. Upper case and lower case letters are considered different.

## 2.3 Keywords

All keywords are case sensitive. The following identifiers are reserved for use as keywords, and may not be used otherwise:

zone	allow
policy	deny
default	on
group	by
principal	when
object	if
alias	else
url	log
email	notify
phone	meta_action
member	system
actions	constraint
action	requirement
number	assertion
string	demand
boolean	consequence
true	
false	

Some keywords, including constraint, requirement, assertion, demand and consequence, are not used in the current design of SPCL. They are reserved for future uses.

## 2.4 Strings (Values)

A string is a sequence of characters surrounded by double quotes ' " '. The meaning of this string value is system dependant. The language itself merely treats it as some mysterious sequence that describes the condition of certain object, or a representation of certain actions to be performed. Escape character is not permitted in the current design.

## 2.5 Numbers (Values)

A number is any real number with an optional sign in front of it. E-notation is not supported in this language. A real number is a sequence of digits with at most one optional decimal point some where in this sequence.

## 2.6 Operators

Operators are further divided into five different sub-types, known as relations, assignment operator, star sign, dot operator and structural operators.

### 2.6.1 Relations

A relation must be one of the following: ">", "<", "<=", ">=", "==", "!=". A relation, along with variables and values, are used for expressing conditions of the environment.

### 2.6.2 Assignment Operator

The assignment operator is a single equal sign '='. It is used to initialize or change the value of certain variables.

### 2.6.3 Star Sign

The star sign is a single star '\*' character that is not surrounded by any double quotes. It is used to express the idea of "everything". Depending on the context of the statement, "everything" might have slightly different meanings. When this token is permitted in a statement, its actual meaning would be further defined in the corresponding section.

### 2.6.4 Dot Operator

The dot operator is a single period '.'. This is used to refer variables of a specific object. See section 6 Object Declaration.

### 2.6.5 Structural Operators

Structural operators serve as the boundary of certain structural blocks and have virtually no meaning in the context of SPCL. These operators include: open parenthesis '(', close parenthesis ')', open brace '{', close brace '}', comma ',', semi-colon ';'. Although no semantic is bound to any of these operators, the exact location on which any of them could be placed is strictly defined by each structural block.

## 3 Syntax notation

The syntax notation used in this manual follows the guideline of Antlr's tool. In general, it is in the form

```
syntax_name : rule_1 | rule_2 | rule_3 ... ;
```

where syntax\_name is a name that represents that particular syntax and all the rules, rule\_1, rule\_2 ... rule\_n, are the possible choices of this syntax. A few other notations are also used in this manual:

```
syntax_name : ( rule_1 ) * ;           // zero or more occurrence
syntax_name : ( rule_1 ) ? ;           // zero or one occurrence
syntax_name : rule_1 rule_2 rule_3 ... ; // concatenation of rules
```

The author believes that the above notations shall suffice in the scope of this manual. If reader is not satisfied with the above description, please refer to [Antlr's homepage](#) for further details.

## 4 Zone

A **zone** is defined to be a finite administrative domain whose name must be agreed upon in between both the administrator and the system. In other words, when the administrator defines a policy on certain resources R under a specific zone Z, the system or the resources of the system, R has to know that it is under that zone named Z. This should not be a problem since the administrator is supposed to be the person who has absolute knowledge of everything under his/her administration.

The syntax of zone declaration is as follow:

```
zone : "zone" ID ';' ;
```

At the beginning of each SPCL file, the above declaration has to be present in order to tell which administrative domain the policy belongs to. **Zone** is the keyword, while ID is the name of the zone that is bound to the specific policy.

## 5 Policy

Conceptually, a policy is a set of rules governing the behavior of a system under a given zone.

In the context of SPCL, policy is the main body of the whole file. For the current design of SPCL, only one policy is allowed per file, i.e. each SPCL file consists of exactly one zone declaration and exactly one policy declaration.

Each policy, in turn, is composed of various sub-components, namely object definition, group definition, principal definition, and default principal definition. These sub-components have to be in the given order. As a result, the syntax of policy becomes:

```
policy : "policy" ID '{'
        default_definition
        (group_definition)*
        (principal_definition)*
        (object_definition)*
        '}' ;
```

**Policy** is the keyword, while ID is the name of this policy. Each sub-component is further defined in subsequent sections. Note that default\_definition is a required component.

## 6 Object Declaration

An SPCL object is an abstract data that represents a resource that is available on the system. With the exception of predefined object "system", all objects must be explicitly declared in the document.

### 6.1 User Defined Objects

The exact syntax of object declaration is as follow:

```
object_definition : "object" ID '{'
                  ( "url" '=' string ';' )?
                  ( variable_declaration )*
                  ( "actions" '{' single_action (single_action)* '}' )?
```

```

        '}' ;
variable_declaration      : ( "number" ID ('=' number)? ';' )
        | ( "string" ID ('=' string)? ';' )
        | ( "boolean" ID ('=' ( "true" | "false" ) )? ';' )
        ;
single_action             : "action" ID '=' string (',' string)* ';'
        ( ID ".meta_action" '=' ( "log" | "notify" ) pID ';' )?
        ;

```

**Object** is a keyword for declaring object while ID is the name that binds to it.

Within the braces, the first option, **url**, is meant to be the identification of the object in the real world.

The second option is an enumeration of all variables that belong to this object. There are three possible types for a variable: **"boolean"**, **"number"** or **"string"**. The value after equal sign, when present, is the initial value of a variable. When referring to variable A of object O, use the ID of O followed by a single dot operator '.' followed by the ID of A. This is exactly the same convention used in many modern object-oriented languages, such as C++/Java.

The third option enumerates all possible actions on this object, using the keyword **actions**. All of the string above complies with the definition stated in section 2.4 Strings (Values). Furthermore, they are treated as regular expressions, for matching rules. If the optional meta\_action exists for a given action, whenever that action is requested, the meta\_action would be executed. See section 11.5.1 Meta Action, for details about meta action.

## 6.2 Predefined Object

There is a predefined object called "system". In the current design of SPCL, there are two predefined variables for object "system", namely state and time. They are both of type string.

The first one (state) is a string description of the current status of the environment. The meaning of this string is system dependent.

The second one (time) is a string representation of current system time in local time zone. The smallest distinction of any two given time is one minute. The time is of the form "HH:MM AP", where HH represents the hour and MM represents the minute. AP could either be "am" or "pm" and must be in lower case. A day starts at time 12:00 am and ends at 11:59 pm.

To refer to one of these predefined variables, use system.variable\_name, where variable\_name is the variable that one is trying to refer to.

## 6.3 Predefined Variable

In addition to predefined objects, there are also predefined variables, that are applicable to every single defined objects. Currently, only state is available. The idea behind this variable is to encapsulate the current status of certain resources. In order to examine this information, one may use objectID.state, where objectID is the object to be examined. There is no need to define this variable prior to usages. Furthermore, if any one is trying to declare a variable with name, state, compiler error shall be generated.

# 7 A Quick Glance at Rule

The complete definition of rule is rather involved. Please refer to section 11 Rule Declaration for a detailed discussion about the syntax of rule.

However, in order for reader to understand the structure of group, principal and default declaration, the author foresee a need to explain briefly the meaning and structure of rules. To present the general idea of rules, the simplest form is provided as follow:

```
rule      :      ("allow"|"deny") (action_list) "on" oID ';' ;
action_list : action (',' action)* ;
action      : string;
```

The above definition says that each rule states whether the administrator ("allow") allows or ("deny") denies certain form of actions (action\_list) on certain objects (oID). Each rule is applicable only to the entity that embraces it, unless overloaded by other optional factors.

All objects must be explicitly declared the document. Also, all actions in the action\_list must match with one of the actions defined in that particular object. If any of the above assertion fails, a compiler-error shall be generated.

Again, though the above discussion is perfectly compatible with SPCL's syntax, it merely exhibits the basic functionality of rules. For further details, please refer to the discussion below in section 11 Rule Declaration.

## 8 Group Declaration

Conceptually, a group binds principals with similar privileges together. It helps the administrator to organize regulations among many users.

In the current definition of SPCL, each principal could belong to any number of groups. Once a group is given a certain amount of privileges or restrictions, all members of that group would be given the privileges.

A group declaration assures the existence of a group and provides it with a unique name. The associated rules for the group are also defined. But, the member that belongs to it is not declared here. For binding members to groups, see section 9 Principal Declaration.

The syntax of a group definition is as follow:

```
group_definition : "group" ID '{'
                  (rule)*
                  '}' ;
```

**group** is the keyword for declaring a new group. The ID following **group** is the name used to refer to this group. Within the body of a group definition, there could be zero or more rules that are applicable to every member of this group. Rule is defined in section 11 Rule Declaration.

## 9 Principal Declaration

A principal is a representation of a user in a system. Each principal could belong to any number of groups discussed in the previous section (Section 8 Group Declaration). The syntax of principal is as follow:

```
principal_definition :  "principal" ID '{'
                        ( "alias" '=' ID ( ',' ID)* ';' )?
                        ( "url"  '=' string ';' )?
                        ( "email" '=' string ';' )?
                        ( "phone" '=' string ';' )?
                        ( "member" '=' gID ( ',' gID )* ';' )?
                        ( rule ) *
                        '}' ;
```

The declaration starts with the keyword **principal**, followed by an identifier that uniquely determines the principal from other entities.

The first line is a list of all optional aliases used in the SPCL. i.e., the administrator could refer to a particular principal using its unique principal ID as well as these aliases. The names used by aliases are also unique identifiers in an SPCL document. As a result, in the context of SPCL, "the name of a principal" implies either the actual ID of the principal or one of the aliases of that particular principal.

The three optional lines followed are meant to be the real world's identity of the principal. For the current version of SPCL, only the keyword **email** and **phone** are permissible. These are used when a meta-action is defined on this principal. See section 11.5.1 Meta-Action, for further details.

The next option, **member**, is the binding in between groups and principals. Each gID is the identifier for a group that is defined in the document. If gID is not found, a compiler error shall be generated.

For the definition of rule, see section 11 Rule Declaration.

## 10 Default Declaration

The default declaration describes the behavior of an unidentified user. It also serves as the final consult whenever related rules cannot be found in both the principal's declaration and all of the related groups' declaration. This is the only required declaration in an SPCL document. The syntax of default declaration is as follow:

```
default_definition :  "default" '{'
                      ( rule ) *
                      '}' ;
```

The declaration of default block starts with keyword default followed by any number of rules surrounded by braces.

## 11 Rule Declaration

A rule in SPCL is a statement that binds principal with certain privileges or restrictions. In other words, a rule says what can be done or can't be done by a person.

## 11.1 Basic Rule

A basic rule is the simplest form of rule that is used by an SPCL document. The syntax is as follow:

```
basic_rule      : ( "allow" | "deny" ) (action_list) "on" oID ;
action_list    : action ( ',' action ) * ;
action         : string ;
```

In the current design of SPCL, **allow** and **deny** are the only keywords for granting access or restricting access on any resources in the system.

Following one of these keywords, is a list of actions separated by comma. Each of these actions is a string, which follows the definition in section 2.4 String (Value), that matches with one of the many actions defined in object oID. See section 6 Object Declaration for action definitions.

Actions stated in a rule are always treated as explicit strings that represent specific actions as opposed to a regular expression in object declaration. All actions must be surrounded by double quotes.

The last part starts with keyword on, followed by oID, where oID is the name of an object declared some where in the document. This object tells where the action is going to be performed on.

(Note: If you compare the definition state here with the definition stated in section 7 A Quick Glance at Rule, you may noticed that a semi-colon is missing. This is because the semi-colon is the terminator of a complete rule while the basic rule is just the beginning of the whole story. Other options discussed in subsequent sections shall be placed before the final terminator. Also note that semi-colon is just one possible way to terminate a rule )

## 11.2 Usage of \*

In the context of rule definition, a star sign '\*' is used as a shorthand for the notion of every possible action. When this operator is included in the rule definition, the syntax is as follow:

```
basic_rule      :      ( "allow" | "deny" ) action_and_target ;
action_and_target : ( ' * ' ( "on" oID ) ? )
                  | (action_list) "on" oID ;
```

When the optional keyword "on" exists and is followed by an object ID (oID), the star sign is a short hand for all possible actions defined in object oID. If the target object is not stated, the star sign means all possible actions on all defined objects.

## 11.3 Optional By-Clause

The first option that follows the basic rule is a by-clause. The syntax is :

```
by_clause :      ( "by" pID ( ',' pID ) * ) ? ;
```

A by\_clause starts with the keyword **by**, followed by a list of names separated by commas. Each of these names refers to a principal defined some where in the document. If the principal is not defined, a compiler-error shall be generated.



Normally, when a rule is defined, it is being imposed onto the entity that embraces the declaration of the rule. If this optional by-clause exists in the rule, the rule is being imposed onto the entity/entities defined in the list instead.

This functionality is only available in the rules used in rule updates. See the details in 11.5 Optional Side-Effects.

If the by-clause exists in any rule other than a rule updates, a compiler-error shall be generated.

## 11.4 Optional When-Clause

Following the optional by-clause is the when-clause. The syntax is as follow:

```
when_cause :      "when" '(' '
                  oID '.' vID relation value
                  ( ',' oID '.' vID relation value ) *
                  ')' ;
value : string | number | "true" | "false" ;
```

A when-clause begins with keyword **when**, followed by an open parenthesis, followed by a list of conditions separated by commas, and is terminated by a close parenthesis. The definition of relation, string and number are stated in section 2 Lexical Conventions. oID is the name of an object that is declared in the document, while vID is the name of a variable that belongs to oID. All conditions are ANDed together to determine the final result. Boolean OR is currently not available in SPCL.

## 11.5 Optional Side-Effects

The last option of a rule is side-effects. There are three possible structures of side-effect, namely meta-action, rule-update, and conditional-side-effect. When side-effect exists, the terminator shall be replaced. As a result, the terminator of a rule would be as follow:

```
terminator :      ';'
                | '{' ( side_effect ) * '}'
                ;

side_effect :      meta_action
                | rule_update
                | conditional_side_effect
                ;
```

The side effects get triggered whenever the rule is being applied to a request. Each of the sub-components is to be discussed immediately in subsequent sections.

### 11.5.1 Meta Action

The idea of meta-action is to provide some kind of mechanism to inform certain party when unexpected events occurs. Currently, only log and notify are supported. The syntax is as follow:

```
meta_action :      ( "log" | "notify" ) pID ';' ;
```

where pID is the name of a principal that has been defined. In order for this function to operate appropriately, the information has to be provided in the principal's definition.

### 11.5.2 Rule Update

In the context of SPCL, updating a rule means introducing a new rule dynamically. These new rules would be placed at the principal precedence level (see 12 Rule Conflicts and Precedence Levels). The syntax of rule\_update is just rule, i.e.:

```
rule_update :    rule;
```

The right hand side, rule, could includes everything that has been discussed in section 11 Rule Declaration, or, to be more precise, the definition that is going to be presented at 11.6 Gathering All Elements Together.

### 11.5.3 Conditional Side Effects

A conditional side effect is just another flexible feature of SPCL. The administrator might decide to have certain side effects based on some conditions other than those that are already matched by the rule. The syntax is as follow:

```
conditional_side_effect :    "if" condition_list '{'
                                ( side_effect ) *
                                '}'
                                ( "else" '{' ( side_effect ) * '}' )?
                                ;
```

Condition\_list has exactly the same format as in when-clause (section 11.4) except that the keyword "when" is missing. Side\_effect has already been defined in section 11.5.

The syntax says a side effect could be composed of nested if-else blocks to determine the exact side effect that is applicable to a given situation.

## 11.6 Gathering All Elements Together

For readers' convenience, the complete definition of rule is duplicated here :

```
rule      :    ("allow" | "deny")
              (
                ( '*' ( "on" oID )? )
                |
                action_list "on" oID
              )
              ( by_clause ) ?
              ( "when" condition_list )?
              ( ';' | '{' ( side_effect )* '}' )
              ;

action_list :    action ( ',' action )* ;
by_clause   :    ("by" pID ( ',' pID)* )? ;
condition_list :
                '('
                oID '.' vID relation value
                ( ',' oID '.' vID relation value ) *
                ')'
                ;

side_effect :    ( ( "log" | "notify" ) pID ';' )
                | ( rule )
```

```

| (      "if" condition_list '{'
          ( side_effect ) *
        '}' "else" '{'
          ( side_effect ) *
        '}'
      )
;

```

## 12 Rule Conflicts and Precedence Levels

Rules defined in different blocks have different precedence levels. The following precedence are defined:

rule level	precedence
principal	20
group	10
default	0

Each rule is associated with a rule level, and its corresponding precedence value. When there is a conflict between two rules, the one with higher precedence value dominates.

If conflicts happens within the same level. The action requested would be rejected and a run-time error would be generated. However, if the conflict is foreseeable during compile time, a compiler-error shall be generated to stop ambiguity instead.

### 12.1 Principal Level

A rule is defined to be at principal level when the rule exists as a direct rule under a `principal_definition` discussed in section 9 (Principal Declaration). In addition, if a rule is generated on demand (see Section 11.5.2 Rule Update). It is also considered to be at principal level.

### 12.2 Group Level

A rule is defined to be at group level when the rule exists as a direct rule under a `group_definition` discussed in section 8 (Group Declaration).

### 12.3 Default Level

A rule is defined to be at default level when the rule exists in `default_definition` discussed in section 10 (Default Declaration).

## 13 Implicit declarations

Implicit declaration is not allowed in all aspects of SPCL. For example, whenever the administrator wants to define a rule on the usage of object A, A must be declared some where in the document. The same restriction applies to action, principal and group.

# 14 Regular Expressions

The regular expressions used in SPCL conform to JAVA's conventions. Please refer to JAVA API for further details on the exact syntax of regular expressions.

# 15 Examples

```

/*****
 *      Example 1.
 *      A simplest SPCL files that
 *      denies all kinds of activities.
 *
 *      Obviously, this is not a very
 *      useful policy. But, it serves
 *      as a hello world program for
 *      SPCL.
 */

zone Restricted_Area;

policy No_Activity_Allowed {
    default {
        deny *;
    }
}

/*****
 *      Example 2.
 *      One principal gets the authority to
 *      perform any actions while every body
 *      else is under strict control
 *
 *      This policy demonstrates some
 *      essential structures of SPCL,
 *      such as declaring objects and
 *      principals.
 */

zone Nazis;

policy Dictatorship {

    default {
        deny *;
    }

    principal Hitler {
        alias    = dictator;
        email    = "hitler@nazis.org";
        allow    *;
    }
}

```

```

principal Assistance {
    allow    "read" on Top_Secret_Documents;

    /* although no side effect is stated here,
       there is a default meta_action defined for
       this action under object Top_Secret_Documents.
       As a result, when the Assistance is trying
       to "reveal" Top_Secret_Documents, the
       system would notify the dictator.
    */
    deny     "reveal" on Top_Secret_Documents;
}

object Top_Secret_Documents {
    actions {
        action a = "reveal";
        a.meta_action = notify dictator;
        action b = "read";
    }
}

```

```

}

```

```

/*****

```

```

*      Example 3.
*
*      Here is a more practical example:
*
*      A webserver www.peterpan.com
*      works in the following manner:
*
*      It limits the number of users
*      to a certain threshold
*
*      The webserver closes during the
*      night for maintenance.
*
*      When the number of requests
*      on TCP port 80 at a particular
*      instance exceeds a certain
*      threshold, notify the network
*      administrator.
*
*      When there are file transfers
*      to or from the webserver,
*      log all the activities.
*/

```

```

zone Internet;

```

```

policy Website_Policy {

    default {

```

```

        deny *;
    }

principal Network_Administrator {
    alias    = admin;
    email    = "admin@peterpan.com";
    phone    = "212-123-4567";
    allow    * on webserver;
}

principal Guest {
    alias    = user;
    allow    "http" on webserver when (
        system.time >= "06:00 am",
        system.time <= "09:00 pm",
        webserver.number_of_connections < 1000
    ) ;
    deny     "http" on webserver
    when ( webserver.number_of_connections >= 1000 ) {
        {
            // maybe there is an attack?!
            if ( webserver.number_of_requests > 50000 ) {
                notify admin;
            }
        }
    }
}

principal Hacker {
    alias    = Trudy, EvilGuy, Unemployed;
    deny     *;
}

principal LogFile {
    url      = "http://www.peterpan.com/cgi-bin/logactivity";
    allow    *;
}

object webserver {
    url      = "http://www.peterpan.com/";

    // these are not set by the policy
    // but would be retrieved from the environment
    number   number_of_connections;
    number   number_of_requests;

    actions {
        action a = "ftp";
        a.meta_action = log LogFile ;
        action b = "http";
    }
}
}

```

# APPENDIX 1 Syntax Summary

```

zone :           "zone" ID ';' ;

policy :         "policy" ID '{'
                  default_definition
                  (group_definition)*
                  (principal_definition)*
                  (object_definition)*
                  '}' ;

default_definition :  "default" '{'
                      ( rule ) *
                      '}' ;

group_definition :   "group" ID '{'
                     (rule)*
                     '}' ;

principal_definition :  "principal" ID '{'
                        ( "alias" '=' ID (',' ID)* ';' )?
                        ( "url" '=' string ';' )?
                        ( "email" '=' string ';' )?
                        ( "phone" '=' string ';' )?
                        ( "member" '=' gID (',' gID)* ';' )?
                        ( rule ) *
                        '}' ;

object_definition    :  "object" ID '{'
                        ( "url" '=' string ';' )?
                        ( variable_declaration ) *
                        ( "actions" '{' single_action (single_action)* '}' )?
                        '}' ;

variable_declaration :  ( "number" ID ( '=' number )? ';' )
                        | ( "string" ID ( '=' string )? ';' )
                        | ( "boolean" ID ( '=' ( "true" | "false" ) )? ';' )
                        ;

single_action        :  "action" ID '=' string (',' string)* ';'
                        ( ID ".meta_action" '=' ( "log" | "notify" ) pID ';' )?
                        ;

rule :              ( "allow" | "deny" )
                    ( '*' ( "on" oID )? )
                    | (action_list "on" oID)
                    )
                    ( by_clause ) ?
                    ( "when" condition_list )?
                    ( ';' | '{' ( side_effect ) * '}' )
                    ;

```

```

action_list :          action ( ',' action ) * ;
by_clause :          ( "by" pID ( ',' pID ) * ) ? ;
condition_list :
    '('
        oID '.' vID relation value
        ( ',' oID '.' vID relation value ) *
    ')'
;

value : string | number | "true" | "false" ;

side_effect :
    ( ( "log" | "notify" ) pID ';' )
    | ( rule )
    | (
        "if" condition_list '{'
            ( side_effect ) *
        '}'
        ( "else" '{' ( side_effect ) * '}' ) ?
    )
;

```

---