CA Configuration Automation®

Blueprint Developer Guide

r12.8

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Document Overview

This document is intended for CA Configuration Automation users responsible for creating or modifying Blueprints. It provides an overview of what Blueprints are and how they are used. The bulk of the document describes the step-by-step procedures associated with Blueprint creation.

Chapter 2: Blueprint Overview

Blueprints are the abstract definitions or *metadata* for a software component. This metadata defines the directives and mechanisms to:

- Detect a software component on a given computer
- Capture file system and database elements of the component
- Express and show inter- and intra-component relationships and dependencies
- Locate, analyze, and manage the configuration information
- Define, execute, and interpret diagnostic macros
- Define recommended, best-practice values for all these elements

CA Configuration Automation presents each Blueprint in a standardized format that simplifies configuration and administration tasks. CA provides a library of predefined Blueprints for commonly used software components. You can also edit existing Blueprints and can create custom Blueprints.

Understanding the Structure and Contents of a Component Blueprint

The ability to view, add, or modify a Component Blueprint or Component Blueprint elements depends on the CA Configuration Automation roles assigned to your user account or group. See the *CA Configuration Automation Implementation Guide* for detailed information about CA Configuration Automation roles and privileges.

All Component Blueprints are presented in a standardized tree view, consisting of the following organizational folder elements:

- Nesting
- Indicators
- Managed
- Parameters
- Configuration
- Diagnostics

- Documentation
- Runtime
- Utilities

Each element is described in a section that follows.

Nesting

The nesting element lets you emphasize the relationship between components. When, for example, a software component uses and depends on several subordinate components, and those components are installed within the primary component's file system root, nesting can be used to enforce the parent-child relationship between them.

Oracle databases, for example, normally install a Java Runtime Engine and an Apache Web server within their installation directory. The relationship between the utility components and the Oracle database is expressed by nesting the JRE and Apache components within the Oracle component.

Indicators

The Indicators primary folder defines where to look for and how to identify a component on a server.

Because component discovery has been extended to servers without CA Configuration Automation Agents installed, and CA Configuration Automation does not have the registry and file systems available for this kind of discovery, you need to define different sets of indicators to find components on servers with installed agents and to find components on servers without installed agents.

- On servers with CCA Agents, component discovery uses file indicators to search the file system for a specific set of files and directories that indicates the presence of a component, and then determines the corresponding root directory under which the managed files for the component are located.

  For Windows servers, you can alternatively define registry entries as indicators. Note that if you use registry entries as indicators, you must specify the component's root directory in the parameters primary folder.

  Note: While both file system and registry indicators can be used on Windows platforms, we recommend that you define one or the other, not both. You can, however, define more than one set of indicator type, which allows multiple pattern matching to extend the same Component Blueprint across multiple platforms. A good strategy for building an indicator set is to define two to three files/directories or registry keys/values that have a known position relative to one another as defined in Depth From Root or Path From Root. Defining too many indicators can produce undesirable and incorrect results.

- On servers without CCA Agents, component discovery uses network probes to scan TCP ports, collect responses from those ports, and compare the responses against expected expressions to determine the type of service active on that port.

Indicators are not always sufficient to determine the existence of installed components or cannot distinguish the differences between two components with similar indicators. The verification directives folder contains directives that are used to discard components that are partially installed, of the wrong version, or not of interest to a particular service.

Note: Because of the order in which verification directives are executed in component discovery, database and configuration file parameters may not be available.

Adding Indicator Elements

To Add a...	Access and Procedure
File Search Option	Click on the files folder, then click the Add Search Options button.
Directory Indicator	Click on the files > / (file root) folder, then click the Add Directory button.
File Indicator	Click on the files > / (file root) folder, then click the Add File button.
Registry Search Option	Click on the registry folder, then click the Add Search Options button.
Registry Key Indicator	Click on the registry > \ (registry root) folder, then click the Add Key button.

Registry Value Indicator	Click on the registry > \ (registry root) folder, then click the Add Value button.
Network Probes	Click on the service folder, then click the Add Network Probe button.
Verification Directive	Click on the verification directives folder, then click the Add Directive button.

Managed

The Managed primary folder defines important files, registry entries, and database elements that are associated with the managed component.

If the managed folder contains no files or registry entries, the application manages all files under the component root directory and registry entries under the registry root. If a component has a limited number of files or registry entries, it makes sense to let all such elements under the root be managed. However, for a complex component with many files, it can be more useful to identify only the files and registry entries on which to focus. Identifying specific files and registry entries lets you refine the view of the managed component.

The File System Overlay and Registry Overlay folders let you (optionally) customize specific file and registry elements. For example, you can assign the categories, filters, or weights, or you can attach notes and rules to an element.

The data folder provides the database connection information and lets you:

- Define a database to reference elsewhere in the Component Blueprint
- Manage the database metadata, including table definitions and indexes

Important! Include any file that is referenced in the configuration, documentation, or run-time primary folders as a managed file.

Adding Managed Elements

To Add a...	Access and Procedure
Directory	Click on the files > $(Root) folder, then click the Add Directory button.
File	Click on the files > $(Root) folder, then click the Add File button.
File System Overlay Directory	Click on the file system overlay > $(Root) folder, then click the Add Directory button.

File System Overlay File	Click on the file system overlay > $(Root) folder, then click the Add File button.
Registry Key	Click on the registry > $(RegistryRoot) folder, then click the Add Key button.
Registry Value	Click on the registry > $(RegistryRoot) folder, then click the Add Value button.
Registry Overlay Key	Click on the registry overlay > $(RegistryRoot) folder, then click the Add Key button.
Registry Overlay Value	Click on the registry overlay > $(RegistryRoot) folder, then click the Add Value button.
Database	Click on the data folder, then click the Add Database button.
Database Table	Click on a database in the data folder, then click the Add Table button.

Parameters

The Parameters primary folder contains a Directives subfolder that defines and displays details that are critical for locating and identifying the component, such as the file system or registry root, component version, vendor, and database connection information.

Discovery determines the file system root and registry root of a Component Blueprint and displays these parameters in the discovered service tree view. Special Version and NameQualifier parameters, if defined in the Component Blueprint, are displayed after the name in the discovered service tree view. The Component Blueprint parameter NameQualifier is defined as $(Product Name) SP$(Service Pack) and parameter Version is defined as $(RegistryRoot)\Windows NT\CurrentVersion\CurrentVersion.

Parameter directives can be used for variable substitution into diagnostics, utilities, and the definition of rules and other parameters. Wherever the string $(ParameterName) is entered as a value, the actual value of that parameter is substituted.

Adding Parameter Elements

To Add a...	**Access and Procedure**
Parameter Directive	Click on the directives folder, then click the Add Directive button.

Configuration

The Configuration primary folder defines how to find and interpret the component configuration information. You can find the configuration information in managed files or databases or you can derive it from the output of executables.

The Structure Classes folder defines how to interpret the configuration files, database queries, and executables. The information includes the semantics of each potential value in the configuration data set:

- Data typing
- Default values
- Enumerated values
- Qualifiers
- Categories
- Filters
- Weights
- Rules

Think of a structure class as the metadata.

The application locates, parses, and interprets for viewing and comparative analysis the files that the Configuration Files folder identifies.

The data folder identifies the queries or stored procedures to run to extract configuration data and how to interpret the results for comparative analysis.

The executables folder defines scripts and directives that can extract the configuration information from a server and how to interpret the results for comparative analysis.

Adding Configuration Elements

To Add a...	Access and Procedure
Structure Classes Class	Click on the structure classes folder, then click the Add Class button.
Structure Classes Parameter	Click on a class under the structure classes folder, then click the Add Parameter button.
File	Click on the files > $(Root) folder, then click the Add File button.

Structure Classes Group	Click on a class under the structure classes folder, then click the Add Group button. You can add additional groups and parameters to a group using the Add Group and Add Parameter buttons that display when you select a group. You can also make a copy of a group and all of its associated parameters using the Group Copy button.
Database	Click on the data folder, then click the Add Database button.
Database Query	Click on a database under the data folder, then click the Add Query button.
Executable Directive	Click on the executables folder, then click the Add Directive button.

Utilities

The Utilities primary folder contains executable files, macros, and scripts that can be used to perform common administrative tasks, for example, programs or scripts that start or stop the component or that provide additional information about a server or service, such as viewing system information, memory statistics, or disk volume statistics.

Adding Utility Elements

To Add a...	Access and Procedure
File	Click the file's $(Root) folder, then click the Add File button.
File Usage	Click a file in the file's $(Root) folder, then click the Add File Usage button.
Macro	Click on the macros folder, then click the Add Macro button.
Macro Step	Click on a macro under the macros folder, then click the Add Step button.

Diagnostics

The Diagnostics primary folder defines executable files and macros used for diagnosing, troubleshooting, and fixing component problems.

Any executable, script, or batch file that can run on a server can be defined here and made available to CA Configuration Automation users with the appropriate access control role. Macros provide a way to include frequently used scripts and troubleshooting tools that can help diagnose problems specific to the servers containing the data being managed by its Component Blueprint.

Adding Diagnostic Elements

To Add a...	Access and Procedure
File	Click on the files > $(Root) folder, then click the Add File button.
File Usage	Click on a file in the files > $(Root) folder, then click the Add File Usage button.
Macro	Click on the macros folder, then click the Add Macro button.
Macro Step	Click on a macro under the macros folder, then click the Add Step button.

Runtime

The Runtime primary folder defines component run-time files and helps you locate and view them quickly. The run-time files (for example, log files) typically change frequently. To exclude the file contents from comparative analysis, define the files in the Runtime primary folder.

Adding Runtime Elements

To Add a...	Access and Procedure
File	Click on the files > $(Root) folder, then click the Add File button.

Documentation

The Documentation primary folder defines how to find the component's managed documentation files (for example, readme files, PDF files, or HTML versions of the product manuals). The Documentation folder also lets you add explicit URL links to additional sources of component information or documentation, such as company intranets or vendor support sites.

Note: Access to referenced URLs is determined by your network configuration.

Adding Documentation Elements

To Add a...	Access and Procedure
File	Click on the files > $(Root) folder, then click the Add File button.

URL	Click on the URLs folder, then click the Add URL button.

Chapter 3: Creating Blueprints

This chapter contains the procedure for creating custom Blueprints. The field descriptions contained in this chapter can be used in conjunction with the Blueprint Element Reference described in the next chapter.

Create Blueprints

CA Configuration Automation lets you build and maintain custom blueprints. Although simple blueprints can be easy to build, detailed and complex blueprints require careful planning and testing.

Follow these steps:

1. Click the Management link, and then click the Blueprints tab.
2. Click the Blueprints link.

   The Blueprints page opens, listing all existing blueprints.

   Select Create Blueprint from the Table Actions drop-down list.
3. Complete the following fields on the Component Blueprint wizard Blueprint page:

   Component Blueprint Name

   Defines a unique name for the blueprint.

   Component Version

   Defines the version (release number) of the software component.

   - When a single blueprint supports all versions of a specific component, set this field to *.*
   - Otherwise, define the specific version or version series supported (for example, 1.2 or 2.*)

   Default: *.*

   Blueprint Version

   Defines the version of the blueprint, in your preferred format.

   Default: 1.0.0

   Discovery

   Specifies whether discovery is enabled or disabled for this blueprint.

   Default: Enabled

Description

Describes the blueprint. The description appears on the blueprint Details page.

Operating System

Specifies the operating system on which the software component runs. The operating system that you select determines whether the blueprint:

- Supports a specific operating system (for example, if the file system structure varies significantly across platforms or relies heavily on the Windows registry).
- Works across multiple operating systems (if the file system paths and configuration parameters can be normalized).
- Specifically supports an F5 BIG-IP load balancer.
- Specifically supports the Cisco routers and network switches that use iOS.

You can always add or remove operating system-specific files later. However, choosing the target operating system lets you start with a more representative base blueprint structure. For example:

- If you select Windows, the wizard automatically creates registry- and registry overlay-related components.
- If you select Any UNIX or another UNIX-based operating system, the wizard does not create the registry- and registry overlay-related components.

Category

Specifies the component category under which the Component Blueprints page lists the new blueprint.

Note: To separate your custom blueprints from predefined blueprints, use the Custom Components category. A separate category lets you find the blueprints easily and avoid accidentally editing the predefined blueprints.

4. Click Next.

The Discovery Methods page appears with the File Indicators element selected in the left pane.

5. Complete the following Search Options fields:

   Search From

   Defines the starting point of the discovery indicator search. If you do not set this value, the search begins at the top of the component file system.

   For Windows, the top of the file system (/) includes all physical drives (C:, D:, and so on).

   Default: / (root)

   Search Depth

   Defines the depth (in folders or directories) of the discovery indicator search.

   Default: 10

   Component Always Found

   Specifies whether the process always considers the component as discovered. If you select Yes, CA Configuration Automation does not search for the file indicators.

   If you define a $(Root) parameter as a constant, the product manages files under the root. Selecting Yes lets you define and manage components in services that cannot or need not be discovered.

   Default: No

6. Click Add Directory/File.

7. Complete the following File Indicators fields in the Add New File pane:

   Name (posix)

   Defines the directory or file to discover. You can use wildcard characters (*) to define the name with the POSIX pattern matching syntax (for example, Agent_*).

   Type

   Specifies whether a new file or a directory is added to the blueprint.

(Optional) Path From Root

Note: Specify either the Path From Root field or the Depth From Root value.

Defines the partial path on which to locate the component root relative to the file indicator path. Define only the directories or folders between the file indicator and the component root (not the entire component path).

For example, the file indicator mx.ini is defined to locate the configuration file C:\Program Files\mx\setup\conf\mx.ini. To define the path C:\Program Files\mx as the component root, define the path from the root as setup\conf.

- The search does not support the full expression match. Use wildcard characters in the partial path for intermediate directories with unknown variable names. For example, *\conf
- You can use the * and ? wildcard characters in the directory names. For example: *conf, \conf*, /*bin, /bin*, /b*n, and /bi?

(Optional) Depth From Root

Note: Specify either the Path From Root field or the Depth From Root value.

Defines the depth (the number of folders or directories) from the search root.

8. Click the Registry Indicators link.

   The left pane displays the Registry Indicators and \HKEY_LOCAL_MACHINE folders. The right pane displays the Search Options fields for the \HKEY_LOCAL_MACHINE folder.

9. Complete the following Search Options fields, and then click Save:

   Search From

   Defines the starting point of the discovery indicator search. If you do not set this value, the search begins at the top of the component file system.

   For Windows, the top of the file system (/) includes all physical drives (C:, D:, and so on).

   Default: \HKEY_LOCAL_MACHINE

   Search Depth

   Defines the depth (in folders or directories) of the discovery indicator search.

   Default: 10

   Component Always Found

   Specifies whether the process always considers the component as discovered. If you select Yes, CA Configuration Automation does not search for the file indicators.

   If you define a $(Root) parameter as a constant, the product manages files under the root. Selecting Yes lets you define and manage components in services that cannot or need not be discovered.

   Default: No

10. Click Add Registry Value/Key.

11. Complete the following Registry Indicators fields in the Add New Registry Value/Key pane:

    Name (posix)

    Defines the directory or file to discover. You can use wildcard characters (*) to define the name with the POSIX pattern matching syntax (for example, Agent_*).

    Type

    Specifies whether a new file or a directory is added to the blueprint.

    Path From Root

    Note: Specify either the Path From Root field or the Depth From Root value.

    Defines the path from the search root.

    Depth From Root

    Note: Specify either the Path From Root field or the Depth From Root value.

    Specifies the depth (the number of folders or directories) from the search root.

12. Click the Network Probes link.

    Note: On the servers without CA Configuration Automation agents, discovery operations can use network probes to:

    - Scan ports
    - Collect the responses from the scanned ports
    - Determine the type of service that is active on a scanned port by comparing the responses against expected expressions

13. Complete the following Network Probes fields:

    Name

    Defines the name of the network probe.

    Primary Ports

    Defines a comma-delimited list of the first port numbers from which to collect responses.

    Alternate Ports

    Defines the port numbers from which to collect responses if the primary ports fail, separated with commas.

    Regex Match

    Defines the attribute that the probe compares to the regular expression directive value. If the value does not match, the directive fails.

    Version Regex

    Defines the regular expression version.

    Protocol

    Specifies whether the probe uses Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).

14. Click Next.

    The left pane displays the Discovery Verification Rules folder appears. The right pane displays the Discovery Verification Rule fields.

    The product runs the verification rules during discovery to verify that the discovered components are correctly identified. In some cases, the file and registry indicators cannot determine the existence of installed components. Similarly, the file and registry are sometimes unable to distinguish between two components with similar indicators. If a verification rule fails, the component discovery fails.

15. Complete the following fields:

Name

Defines the Discovery Verification Rule name.

Description

Describes the purpose of the rule.

Directive Type

Specifies the type of directives the product uses to extract values from the elements that a service manages or to retrieve values from managed servers. The Directive Type drop-down list contains the following options:

Constant

Defines a fixed value that the product uses to substitute variables or to construct complex strings. This directive type has the following common uses:

- Defining the fixed Root and RegistryRoot parameters when there is no ambiguity about the installed location of a component.
- Assembling the variables and fixed text into a complex string, combining data from multiple sources.
- Exposing component constants, such as the vendor name, as parameters.

Database Query

Queries a database on a managed server, and (optionally) extracts a value from the data retrieved. This directive type has the following common uses:

- Running a query to extract a column value from a row in the database.
- Running a stored procedure to change the managed server or to extract a value.
- Running a query to extract a result set and display it in tabular form in a macro step directive.

Get File

Retrieves the contents of a specific file (when the location of that file is known), and then filters the contents to define a directive value. This directive type has the following common uses:

- Extracting the parameters from unstructured files by fetching the file contents and filtering them with a regular expression.
- Retrieving log files for display, storage, and import and export in macro step directives.

Get SNMP

Retrieves the value at a management information base (MIB) address from an SNMP agent. (Optionally) Filters the retrieved value to set the directive value.

Match File Name

Lists all files and directories at a designated location in a managed server file system. (Optionally) Filters the list to extract the directive value.

Match Registry Name (Windows only)

Lists all registry key and registry value names at a location in the registry tree. (Optionally) Filters the list to extract the directive value.

Match Registry Name Get Data (Windows only)

Lists all registry key and registry value names at a location in the registry tree. (Optionally) Retrieves the data value of the selected names as the directive value.

Network Probe

Defines the parameters of a TCP socket opened to a remote service server and port. An optional probe can be sent to the port, and a response that is collected from the socket as the directive value.

Registry (Windows only)

Retrieves the data value that is associated with a registry key or registry value. (Optionally) Filters the result to define the directive value.

Remote Execution

Runs a command or script on a managed server. The product captures output from the command and returns it as the directive value. A regular expression commonly filters the value to extract a concise value from verbose command output. This directive type has the following common uses:

- Accessing the configuration information that is only available from output (for example, operating system configurations).
- Accessing transient data such as memory, network, or CPU statistics.
- Accessing custom scripts and tools and importing their output to the blueprint.
- Updating a managed server by running utilities and scripts.

Web Service Call

Retrieves the application configuration data that is exposed as web services. The directive queries the web services and parses the returned data into configuration variables. The directive supports the configuration executables that can retrieve and store the configuration data.

Provide the WSDL URL where the web service is running. Web Services Description Language (WSDL) is an XML-based language that provides a model for describing web services.

Stage

Specifies when verification directives run during discovery, relative to when the component parameter directives run:

Late

The verification directive needs a Component parameter value (for the variable substitution). The Component parameter value ensures that the substituted variables have the values that are required for the verification.

Early

The verification directive does not depend on the value of any Component parameters.

Default: Late

Value

Specifies fixed string or a string that contains one or more variable substitutions.

- If the directive resolves to a string with a length greater than 0, that string becomes the directive value.
- If the string length is 0 or undefined, the product considers the directive to be without a value and uses any specified default value instead.

You can define a constant value as blank (one or more spaces). The blank value appears as no value on the user interface, but CA Configuration Automation considers it a valid value.

Examples:

- Variable substitution with a fixed default:

  `$(VariableName)`

- Multiple substitutions in fixed text:

  `C:\$(V)\$(V2)\file.xml`

You can also use the following variable syntax:

$(*parameter_name*)

Specifies normal component variable substitution.

$(#*parameter_name*)

References a parameter in the parent component.

$(@*global_variable_name*)

References a global variable.

Regex

Defines a regular expression that describes a set of strings. If the variable substitution is used to define a constant value, you can use a regular expression to filter the value. You can also use the regular expression like a conditional expression to test the value and return it in one of the following ways:

- As paren(0) if the value matches the expression.
- As no value if the value does not match the expression.

Paren

Defines the subexpression that the product returns on a match if you define a regular expression (regex) that includes a parenthesized subexpression. The product uses the default value if the regular expression contains no parenthesized subexpressions.

Default: 0

Regex Match

Defines the attribute that the product compares to the regular expression directive value. If the value does not match, the directive fails.

Must Equal

Defines the attribute to which the product compares the directive value. If the values are not equal, the directive fails.

Value Case

Specifies whether the product considers the case of the letters for matches.

Default: Case Sensitive

Note: The default (Case Sensitive) option does *not* consider Agent_conf and agent_conf to be a match. The Case Insensitive option considers Agent_conf and agent_conf to be a match.

Translate

Defines the translation to apply to the directive output. If no translation matches, the product retains the original directive result.

The Directive values that are extracted from configuration files and the registry are often cryptic strings or integers belonging to an enumeration. Each value corresponds to a configuration state, and the value indicates the interpretation. To be clear when displaying these values in the blueprint, they can be translated to meaningful strings.

Define a mapping of values in the CCA Database that references the translation name and triggers the translation of *from values* to *to values*. The following example is the mapping for a translation named $CCTranslation$__IIS SERVER STATE:

```
<BlueprintTranslation name="IIS server state"
coh_name="IIS_SERVER_STATE" coh_id="23501"
created_by="system_user">
   <BlueprintTranslationEntry translate_from="2"
translate_to="Running" />
   <BlueprintTranslationEntry translate_from="4"
translate_to="Stopped" />
   <BlueprintTranslationEntry translate_from="6"
translate_to="Paused" />
</BlueprintTranslation>
```

Translations are not associated with a specific Blueprint—Use name as the reference name from any blueprint. To avoid conflict with other translations, ensure that each translation has a unique name.

Note: CA Configuration Automation does not currently include the user interface support for defining translation tables. Use the CA Configuration Automation data load format in the example to create files and then use the data loader utility to load them to the database.

Transform

Defines the XSL transform with which to filter a returned XML-formatted directive value. The transformed value replaces the original returned value and can be either a new XML value or any other text that the transformation generates.

To run an XSL transform, specify the location of an XSL file to apply to the directive value. The transform location can be a URL (file, http, or other) that is visible from the server, or a file name in the server CLASSPATH. The product retrieves the transform and applies it to the directive value to produce a new value for the directive.

Insert

Defines the value of a directive that is inserted in another string to produce the desired result for display or for a subsequent operation.

Any string with $(VALUE) (all uppercase letters) in it can be the insertion location. The directive value replaces $(VALUE) to produce the filtered result. For example, if the initial directive value is 3 and the insertion string is 1.$(VALUE).export, the filtered directive value becomes 1.3.export.

Modifier

Specifies either the Does Host Use Agent or Is Alterpoint Device.

Modifier Parameters

Defines the parameters to apply to the specified modifier.

16. Click Next.

The product saves the Discovery Verification Rule values, and then displays the File Management page. The left pane displays the $(Root) folder, and the right pane displays the File Management Options. The File Management page links the following pages:

- File Filters and Attributes
- Registry Management
- Registry Filters and Attributes
- Data Management

These pages let you define important file attributes, registry entries, and database elements that are associated with this managed component.

If no files or registry entries are defined, the product manages all files under the component root directory and registry entries under the registry. If a component has a limited number of files or registry entries, allow all files and registry entries under the root to manage them. However, for a complex component with many files, specify only the important directories, files, and registry entries on which to focus. Identifying specific files and registry entries from the File Management page lets you refine the managed component view.

17. Complete the following fields on the File Management page:

    Managed Depth

    Defines how many directory levels below the file system root that the discovery operation checks.

    Retrieve all files

    Specifies how many files the discovery operation retrieves.

    Selected: The discovery operation retrieves all files.

    Cleared: The discovery operation retrieves up to the number of files that the Maximum Files field specifies.

    Maximum Files

    Defines the maximum number of files that the discovery operation retrieves.

18. (Optional) Click Add Directory or Add File (or both), enter a name and description, and then click Save.

    The product adds the directory or file below the $(Root) directory in the File Management pane.

19. (Optional) Select a node in the left pane and repeat Step 18 to create subdirectories and other nested files.

    The product adds the directory or file below the selected node in the File Management pane.

20. Click the File Filters and Attributes link.

    The left pane displays the directory and file structure that you created in Steps 18 and 19 below the $(Root) folder.

    Follow these steps:

    a. Select the $(Root) folder.

       The Precedence table lists the directories in $(Root).

    b. (Optional) Select a column, and then select Move Up or Move Down from the Select Actions drop-down list to change the directory precedence.

       Consider order when you apply meta-links to File Structure Classes (descriptions, rules, filters, and categories) on the parsed data (parameters and groups) and overlays. The Change Detection, the Compare, and the Rule Compliance operations consider the order to process the content to match correct values.

       For example, consider the following parameter definitions in a Blueprint File Structure Class, each with its own rules, category filters, and weights:

       ab.*

       a.*

       .*

       - All parameters that start with *a* get their respective filters.
       - All parameters that start with *ab* get their respective filters. In this case, the product overrides the previous *a.*.*
       - All parameters that do not start with *a* or *ab* get the respective filters that .* specifies.

       Therefore, define more specific parameters first, and then define more generic parameters.

    c. (Optional) Repeat Step b for the files that the Files table lists.

    d. Assign a category or create a filter for the selected directory or file.

    e. Click a file or directory, and then double-click one or more options in the Available Categories column.

       The product adds the selected options to the Selected Categories column.

    f. Double-click one or more options in the Available Filters column.

       The product adds the selected options to the Selected Filters column.

    g. Select the Weight for the file or directory, and then click Save.

h. Click the Rules tab.

The Rules tab defines rules that constrain file and directory values in the Managed - File System overlay. The rules include both the explicit constraint rules that you create and predefined, implicit constraint rules. For example, if you specify a value or data type for an element, CA Configuration Automation automatically creates an implicit Check Default or Verify Data Type rule.

i. Complete the following fields:

Name

Defines the rule name.

Description

Describes the purpose of the rule.

Documentation URL

Specifies a URL where the rules documentation is located.

Constraint Type

Specifies either the File or Directory constraint type to which the rule applies.

File

- File Modification Date
- File Owner
- File Permissions
- File Size
- File Version
- Product Version

Directory

- Bytes
- Depth
- Directory Modification Date
- Directory Must Exist
- Directory Owner
- Directory Permissions
- File Must Exist
- Number of Directories
- Number of Files

Operation

Specifies the operation that the rule for the selected Constraint Type uses.

- = (equals)
- != (not equal)
- = (ignore case)
- != (do not ignore case)
- > (greater than)
- >= (greater than equal)
- < (less than)
- =< (less than equal)
- Integer Range (inclusive)
- Must Match (regex)
- Must Match (regex, ignore case)
- Must Not Match (regex).

Value

Specifies the value which is taken as the reference for the Operation. The Constraint Type, Operation, and Value define a criteria for a rule.

Disable

Specifies whether the rule is enabled (selected) or disabled (cleared).

On Failure

Defines the string that the product writes to the Rule Compliance results when a compliance operation that uses this blueprint fails the rule.

Severity

Specifies which error level determines rule failure.

- Information
- Warning
- Error
- Critical

21. Click the Registry Management link.

22. Complete the following Registry Management fields, and then click Save:

    Managed Depth

    Defines how many directory levels the discovery operation verifies below the file system root.

    Retrieve all files

    Specifies how many elements the discovery operation retrieves.

    Selected: The operation retrieves all elements.

    Cleared: The operation retrieves elements up to the number that is specified in the Maximum Files field.

    Maximum Files

    Defines the maximum number of elements that the discovery operation retrieves.

23. (Optional) Click Add Key or Add File (or both), enter a name and description, and then click Save.

    The product adds the key or file below the $(Root) directory in the Registry Management pane.

24. (Optional) Select a node in the left pane and repeat Step 23 to create other nested keys and values.

    The product adds the key or value below the selected node in the Registry Management pane.

25. Click the Registry Filters and Attributes link.

    The left pane displays the $(Root) folder and the right pane displays the Precedence tab. The page shows existing keys and values in the corresponding tables.

    a. (Optional) Select a column by which to set the directory precedence, and then select Move Up or Move down from the Select Actions drop-down list.

       The product reorders the directories. The importance of sequencing keys and values is similar to the sequencing described in Step 20.

    b. Click Add Key.

    c. Enter a key name and description.

    d. Double-click one or more options in the Available Categories column. The product adds them to the Selected Categories column.

    e. Double-click one or more options in the Available Filters column. The product adds them to the Selected Filters column.

    f. Select the key weight and whether the filter is recursive.

g. Complete the following fields, and then click Save:

Default

Specifies the default value for the registry key.

Interpret As

Specifies how the product interprets the key:

- Database Name
- Database Table
- Host Name and Port
- Host Name or IP Address
- JDBC URL
- TCP Port Number
- URL
- Web Service URL

Relationship Key

Specifies whether the key is a relationship key type.

Relationship Type

Specifies one of the following relationship types:

Communicates With

Establishes the relationship between an application and a database server.

Manages

Establishes the relationship between a server that manages another server. For example, select Manages to view the relationship between a VMware vCenter Server that manages a VMware ESX host.

Hosts

Establishes the relationship between a server that hosts another server. For example, select Hosts to view the relationship between a VMware ESX host that hosts a VMware virtual machine.

Uses

Establishes the relationship between servers.

Visibility

Specifies whether the value is shown or hidden.

Interpreted Server

Specifies the target server information available in any other component parameters. Use the variable substitution method to define the parameter.

Interpreted Instance

Specifies the target database instance or application instance available in any other component parameters. Use variable substitution methods to define the parameter.

Interpreted Application

Specifies the target application information available in any other parameters. Use the variable substitution method to define the parameter.

The product adds the key below the $(RegistryRoot) directory in the Registry Management pane.

a. In the left pane, select the key to which to assign a value.

b. In the right pane, click Add Value.

c. Enter a name and description.

d. Double-click one or more options in the Available Categories column to add them to the Selected Categories column.

e. Double-click one or more options in the Available Filters column to add them to the Selected Filters column.

f. Select the Weight and whether the filter is recursive.

g. Enter the following Value Details, then click Save:

Default

Specifies the default value for this registry key.

Interpret As

Specifies that the key is interpreted as one of the following: Database Name, Database Table, Host Name and Port, Host Name or IP Address, JDBC URL, TCP Port Number, URL, or Web Service URL.

Relationship Key

Specifies whether the key is a relationship key type.

Relationship Type

Specifies one of the following relationship types:

Communicates With

Establishes the relationship between an application and a database server.

Manages

Establishes the relationship between a server that manages another server. For example, select Manages to view the relationship between a VMware vCenter Server that manages a VMware ESX host.

Hosts

Establishes the relationship between a server that hosts another server. For example, select Hosts to view the relationship between a VMware ESX host that hosts a VMware virtual machine.

Uses

Establishes the relationship between servers.

Visibility

Specifies whether the value is shown or hidden.

Interpreted Server

Specifies the target database instance or application instance available in any other component parameters. Use variable substitution methods to define the parameter.

Interpreted Instance

Specifies the target database instance or application instance available in any other component parameters. Use variable substitution methods to define the parameter.

Interpreted Application

Specifies the target application information available in any other parameters. Use the variable substitution method to define the parameter.

The key is added below the $(RegistryRoot) directory in the Registry Management pane.

1. Click the Data Management link.

   The Data Management folder appears in the left pane. The Database page appears in the right pane.

2. Enter the following information in the appropriate fields to define a database in the Blueprint, then click Save:

   Name

   Specifies the database name

   Description

   Describes the purpose of the database.

   Database

   Specifies the database access description.

   User

   Specifies a user name that can access the database.

   Password

   Specifies the password for the user account.

   Server

   Specifies the server where the database is installed.

   Port

   Specifies the listening port of database host server..

   DB Type

   Specifies one of the following: DEFAULT_CONTEXT, SQL_SERVER_CONTEXT, ORACLE_CONTEXT, INFORMIX_CONTEXT, DB2_CONTEXT, MYSQL_CONTEXT, POSTGRES_CONTEXT, HSQLDB_CONTEXT, ORACLE9_CONTEXT, ODBC, CLOUDSCAPE, ORACLE10_CONTEXT, SYBASE11_CONTEXT, INGRES_CONTEXT, SQL_SERVER2K5_CONTEXT, JAVA_DB_CONTEXT, SYBASE15_CONTEXT, ORACLE11_CONTEXT, or SQL_SERVER2K8_CONTEXT.

   Environment

   Specifies the environment variables that are set before the command is executed.

   The remote executables are invoked within the environment of the CA Configuration Automation Agent. If the Agent does not have the environment setup required to run a command, define necessary environment variables as a comma-separated list of name-value pairs in the form name=value (for example, *<var_name>*=*<value>*,*<var_name2>*=*<value2>*).

   The agent defines these environment variables before the command invocation. The Variable substitution method is supported.

   The database appears in the Data Management pane.

3. Click Next

   The Component Parameters and Variables page appears.

4. Enter the appropriate information in the following fields, then click Save:

Name

Specifies the parameter name. Parameter substitution expressions have the form: $(*VariableName*).

Where the *VariableName* is the name of a discovery parameter that is defined in the component. By default, the name is case-sensitive, so it must match che parameter name exactly. The parameter expressions can be embedded within string literals or used on their own. You an define recursive multiple substitutions in the same expression. Fo example, the value of a substitution may be a string in the parameter expression, and if so, it is recursively evaluated.

Example:

Given Discovery parameters with the following values:

```
User=info
Domain=ca.com
v1=$(User)
v2=$(Domain)
```

The following parameter substitution expression:

```
$(User)@$(Domain) [$(v1) at $(v2)]
```

would return the following results after evaluation:

```
info@ca.com [info at ca.com]
```

Description

Describes the purpose of the parameter.

Folder

Specifies the location of the parameter.

Selected Categories

Specifies the following categories into which you want to organize the elements:

Administration

Specifies administrative settings that have to do with the general availability and management of the component. Examples of administrative settings would be how to back up, when to delete a cache, or how many times to retry something.

Configuration

Specifies configuration settings for the component, except for the Administration, Log and Debug, Network, or Performance. Examples of configuration settings would be a component alias name or default web page.

Documentation

Specifies the elements that document the component behavior or act as a guide to users. For example manuals, Readme files, FAQs, or Online help pages.

Log And Debug

Specifies the elements that lets you set log locations, log levels, debug output, or diagnostic variable types.

Network

Specifies the elements that represent or indicate a network-related setting for the component. An example would be the port settings for SNMP. If an element can be categorized as both Network and Security (for example, enabling LDAP Authentication), use Security as the category.

Other

CA internal use only.

Performance

Specifies the settings that impact the performance and are generally a specific subset of configuration parameters. For example, number of threads or number of concurrent users.

Product Info

Specifies general (static) information about the product. Examples of static component information would be licensing, installation location, vendor, or module name.

Resources

Specifies component resources. Examples of component resources would be storage, memory and cache allocation or size, or CPU.

Note: The static resource category is different from the Transient category, which relates more to real-time information.

Security

Specifies the elements that represent security-related settings and are generally a specific subset of configuration parameters. For example authentication types, enabling authentication, encryption settings, directory browsing, SSL, or HTTPS.

Transient

Specifies elements that change with some regularity. For example, server states (for example, up, down, running, or stopped), current number of connected clients, current number of threads, or current disk utilization.

Versioning and Patches

Specifies the elements that indicate the version or patch levels of the product.

Double-click a category in the Available Categories column to move it to the Selected Categories column.

Selected Filters

Specifies the following elements that are excluded from the Compare operations:

Component Specific

Identifies an element that is specific to a single component instance, for example, installation root and Service Server Name. The purpose is to identify and exclude certain component-specific elements, which are already known to be different, from Change Detection operations and results.

Service Specific

Identifies an element that is specific to a service. For example server names and installation roots. The purpose is to identify and exclude certain service-specific elements, which are already known to be different, from Change Detection operations and results.

Server Specific

Identifies an element that is specific to a single server. For example, Server name and IP address. The purpose is to identify and exclude certain server-specific elements, which are already known to be different, from Change Detection operations and results.

Specifies the following elements that are excluded from the Change Detection and Rule Compliance operations:

Never Run Change Detection

Identifies an element to permanently excluded from all types of Change Detection operations and results. A temporary directory, log files in the managed folder, or anything that is known to be transient are examples of elements that you want to identify and always exclude from Change Detection operations and results.

Never Run Rule Compliance

Identifies an element to permanently excluded from all types of Rule Compliance operations and results due to their inconsequential or variable nature. A temporary directory, a known old configuration file, a template, or an example file are the elements to exclude from the Rule Compliance operations and results.

Time Variant

Identifies an element that is known to change over time, but not necessarily across servers or across services. For example, log files, process start times, and registry event counters. The purpose is to identify and exclude parameters that are time variant, known to be different, from Change Detection operations and results.

Double-click a filter in the Available Filters column to move it to the Selected Filters column.

Weight

Specifies the relative importance of an element using Low, Medium, or High. Unweighted elements (no weight assigned) are considered Medium.

Directive Type

Specifies the type of directives that are used to extract values from either elements that are managed within a service or to retrieve values from managed servers. The Directive Type drop-down list contains the following options:

Constant

Defines a fixed value that can be subsequently used for variable substitution or for complex string construction. Common uses for this directive type include:

- Defining fixed Root and the RegistryRoot parameters when there is no ambiguity about the installed location of a component.
- Assembling that variables and fixed text into a complex string, combining data from multiple sources.
- Exposing component constants as parameters. For example, the Vendor name.

Configuration

Retrieves the value of a configuration parameter from a parsed configuration file or from a configuration executable file.

Database Querey

Executes a query against a database on a managed server, and optionally filters the output to extract a value from the retrieved data. Common uses for this directive type include:

- Executing a query to extract a column value from a row in the database.
- Executing a stored procedure to change the managed server or to extract a value.
- Executing a query to extract a result set and display it in tabular form in a macro step directive.

Get File

Retrieves the contents of a specific file (when the location of that file is known), and filters it to define a directive value. The common uses for this directive type include:

- Extracting the parameters from unstructured files by fetching the file contents and filtering them with a regular expression.
- Retrieving log files for display, storage, and import/export within macro step directives.

Get LDAP

Retrieves named data sets from a Directory Server, and optionally filters the output to extract the directive value.

Get SNMP

Retrieves the value at a management information base (MIB) address from an SNMP agent, and optionally filters the retrieved value to set the directive value.

Match File Name

Lists all files and directories at a designated location in a managed server file system, and optionally filters the list to extract the directive value.

Match Registry Name (Windows only)

Lists all registry key and registry value names at a location in the registry tree, and optionally filters the list to extract the directive value.

Match Registry Name Get Data (Windows only)

Lists all registry key and registry value names at a location in the registry tree.It optionally retrieves the data value of the selected names as the directive value.

Network Probe

Defines the parameters of a TCP socket opened to a remote service server and port. An optional probe can be sent to the port, and a response is collected from the socket as the directive value.

Registry (Windows only)

Retrieves the data value that is associated with a registry key or registry value, and optionally filters the result to define the directive value.

Remote Execution

Runs a command or script on a managed server. The output from the command is captured and returned as the directive value. The regular expression filters the value is commonly to extract a concise value from verbose command output.

The common uses for this directive type include:

- Accessing the configuration information that is only available from executable output. Operating system configurations are commonly captured this way.
- Accessing transient data, like memory, network, or CPU statistics.
- Accessing custom scripts and tools that are developed by operations staff and importing their output into the blueprint.
- Changing the managed server by executing utility programs and scripts.

User Input

Prompts the user for the directive value.

Default

Specifies a fixed string or a string containing one or more variable substitutions. If the directive is run and no Value can be determined, the Default value is used in place of an undefined or zero length Value.

Persistence

Specifies the persistence of a parameter within a service.

- Always Persist—The parameter is saved and displays in the service tree view. The default is Always Persist.
- Never Persist—The parameter is used only for Discovery purposes and is not saved.
- Persist If Not Null—The parameter is saved and displays in the service tree view if it has a value.

Visibility

Specifies the visibility of an element and the element value.

- Show Value means that the parameter value displays normally. The default is Show Value.
- Hide Value means that the parameter value does not display and instead show as *******. Hide the parameters if their values are sensitive (such as passwords) or if the values are binary or too long and cannot be displayed. The hidden parameters are encrypted in the database and are not displayed oor viewable in the UI.

  Note: To ensure the security of the contents of this field, enter the value again if you decide at later time to show the value.
- Hide Element lets you hide a parameter. Hide the elements if you want the element value available for the variable substitution, and not view the element displayed in the service tree view.

Value Case

Specifies whether CA Configuration Automation ignores text case differences of values. Case Sensitive (the default) means CA Configuration Automation identifies and reports all differences. Case Insensitive means text case differences are ignored.

Interpret As

Provides the format of the configuration parameter string that an associated component can use. CA Configuration Automation inspects the interpreted parameter values using context-sensitive parsers, which allow the extraction of multiple sub-values from within complex parameter strings.

Default: Blank (no interpretation).

Relationship Key

Specifies that CA Configuration Automation uses the Interpret As field to determine and assign relationships if this field is set to Yes. Not all interpretations defines the relationships.

Default: No.

Important! In order to establish relationships, define a proper value in Interpret As field, and set the Relationship field to Yes.

Relationship Type

Specifies one of the following relationship types:

Communicates With

Establishes the relationship between an application and a database server.

Manages

Establishes the relationship between a server that manages another server. For example, select Manages to view the relationship between a VMware vCenter Server that manages a VMware ESX host.

Hosts

Establishes the relationship between a server that hosts another server. For example, select Hosts to view the relationship between a VMware ESX host that hosts a VMware virtual machine.

Uses

Establishes the relationship between servers.

Interpreted Server Name

Specifies the target server information available in any other component parameters. Use the variable substitution method to define the parameter.

Interpreted Instance Name

Specifies the target database instance or application instance available in any other component parameters. Use variable substitution methods to define the parameter.

Interpreted Application Name

Specifies the target application information available in any other parameters. Use the variable substitution method to define the parameter.

Translate

Specifies the translation that is applied to the directive output. If no translation matches, then the original directive result is retained.

The Directive values extracted from configuration files and the registry are often cryptic strings or integers belonging to an enumeration. Each value corresponds to a configuration state, but the value does not the interpretation. To be clear when displaying these values within the Blueprint, they can be translaten to meaningful strings.

Define a mapping of values in the CCA Database that references the translation name and triggers the translation of from values to to values. The following example is the mapping for a translation called $CCTranslation$__IIS SERVER STATE:

```
<BlueprintTranslation name="IIS server state" coh_name="IIS_SERVER_STATE"
coh_id="23501" created_by="system_user">
   <BlueprintTranslationEntry translate_from="2" translate_to="Running" />
   <BlueprintTranslationEntry translate_from="4" translate_to="Stopped" />
   <BlueprintTranslationEntry translate_from="6" translate_to="Paused" />
</BlueprintTranslation>
```

Translations are not associated with a particular Blueprint—referenced the name from any Blueprint. Ensure each translation has a unique name to avoid conflict with other translations.

Note: Currently, there is no support for the definition of translation tables within the CA Configuration Automation user interface. Files in CA Configuration Automation data load format (shown in the previous example) must be created and loaded to the database using the data loader utility.

Transform

Specifies the XSL transform that is used to filter a returned XML-formatted directive value. The transformed value replaces the original returned value and can be either a new XML value or any other text that the transform generates.

To execute an XSL transform, specify the location of an XSL file to be applied to the directive value. The transform location can be a URL (file, http, or other) visible from the server, or a file name that exists within the server CLASSPATH. The transform is retrieved and applied to the directive value to produce a new value for the directive.

Insert

Specifies the value of a directive that is inserted within another string to produce the desired result for display, or for a subsequent operation.

Any string with $(VALUE) (all capital letters) within it can serve as the insertion location. The directive value replaces the $(VALUE) to produce the filtered result. For example, if the initial directive value is 3 and the insertion string is 1.$(VALUE).export, the filtered directive value becomes 1.3.export.

Modifier

Specifies Does Host Use Agent or Is Alterpoint Device.

Modifier Parameters

Specifies the parameters that is applied to the modifier.

The settings are saved.

1. Click Next.

The Configuration - File Parsing page appears.

2. Complete the following fields on the File Parsing page, and then click the Configuration Executables link (top of the File Parsing page):

   Name (regex)

   Specifies the name of the configuration file that is located and parsed during discovery. You can use regular expressions (for example ^ or $) to define the name using regex pattern matching syntax.

   Display Name

   Specifies the name of the configuration file displayed after discovery.

   Description

   Describes the purpose of the configuration file.

   File Type

   Specifies the type of configuration file (for example, text, binary, or log)

3. On the Configuration Executables page, complete the following fields, and then click Save:

   Name

   Specifies the name of the directive that appears in the Configuration folder under Executables. The name must be unique in the Executables folder.

   Description

   Optional text describing the purpose and origin of the executable directive.

   This description appears as a tool-tip over the executable directive name in the Blueprint and the parameter name in the tree view.

   Directive Type

   Specifies the type of directive used to obtain the executable parameter value. All directives return a value and a boolean result. The value appears next to the executable parameter name when the discovered parameter appears in the service tree view.

   Default: Constant

   The Configuration directive type extracts a value from a managed configuration file.

File

Specifies the name of a configuration file.

The file must be in either the configuration, files folder, or the configuration, executables folder. You can specify a specific file name or a regular expression. If you specify a regular expression, CA Configuration Automation selects all files in the configuration folder that match. If more than one file matches, CA Configuration Automation selects the file closest to the root of the managed file system.

Variable substitution is allowed.

Path

Specifies the path to the configuration file. No wildcards or pattern matching are supported, but variable substitution is allowed. The path can either be absolute (for example, $(Root)/conf) or relative to the root of the managed file system (for example, /conf).

If the configuration file is at the root of the configuration file system, Path can be defined as $(Root), or can be undefined.

Parameter

Specifies the name of the configuration parameter in the file. No wildcards or pattern matching are supported, but variable substitution is allowed.

The Parameter attribute can be undefined if the value belongs to a Group file block. In that case, specify only the Group Path attribute.

Regex

Specifies that if the named value is found, filter it in using a regular expression.

Paren

Specifies that if a regular expression is defined and includes a parenthesized sub-expression, the Paren value indicates which sub-expression should be returned on match.

Default: 0 (if no Paren is specified)

Group Path

Specifies hierarchically organized configuration files use the Group Path value within the file to the named Parameter and is specified like a partial file system path (for example, a/b/c).

Often the same name is used many times within a single file, so a unique Group Path must be specified to identify one particular value from the many matching names.

For example, an XML configuration file can have the following format, where the tag server and attributes buildDate and type occur many times:

```
<configuration>
  <server name= “wxp123”>
    <setup buildDate=”10/30/2003” type=”webserver”/>
  </server>
  <server name=”wxp123” auxiliary=”true”>
    <setup buildDate=”09/02/2002” type=”webserver”/>
  </server>
  <server name=”wxp124”>
    <setup buildDate=”10/29/2003” type=”dataserver”/>
  </server>
</configuration>
```

The elements of a Group Path can be qualified by a comma-separated list of names, or names and values, to identify children that must match for the path to be followed.

For example, to extract the value of buildDate for server wxp124, specify the following:

- Parameter: buildDate
- Group Path: configuration/server,name=wxp124/setup

To extract the buildDate from the wxp123 server’s auxiliary data, only the name is required because it is the only server tag with an auxiliary attribute:

- Parameter: buildDate
- Group Path: configuration/server,auxiliary/setup

Any or all of the elements in a Group Path can be qualified and more than one qualifier can be specified for each element. If more than one qualifier is specified, both qualifiers must match to select a path. Variable substitution is allowed.

If the configuration file has been assigned a structure class, and the structure class defines group blocks qualifiers, an alternate syntax is allowed.

For group blocks, the qualifier value can be appended (within parentheses) to the block name.

For example, to extract the value of buildDate from server wxp124, where the attribute name has been designated a Qualifier Child, specify the following:

- Parameter: buildDate
- Group Path: configuration/server(wxp124)/setup

The value in parentheses matches the qualifier of the named group block. The qualifier is defined by either a Qualifier Child value or a constant Qualifier in the structure class or can be the Value of the group.

Multiple paths can be specified if it is uncertain where a parameter resides within the file. Specify two or more alternate paths separated by a pipe (|). The paths are tried one at a time, from left to right until a match is found. For example:

- Parameter: buildDate
- GroupPath: configuration/server(abc)/setup|configuration/server(xyz)/setup

Empty Is Null

Specifies that iIf the named configuration parameter exists in a file, but has no value, CA Configuration Automation defaults the value to an empty (zero length) string. The Empty Is Null field lets you override this default behavior and make a distinction between a directive returning an empty value and a directive returning no value at all.

- If Empty Is Null is set to No and a default value is defined in the Default field, the value is not be used when the value is empty.
- If Empty Is Null is set to Yes and a default value is defined in the Default field, the value is used when the value is empty.

Default: No

Default

Specifies a fixed string or a string containing one or more variable substitutions. If the directive is run and no value can be determined, the value in the Default field is used in place of an undefined or zero length value.

If the value for a directive can be set by a Component Blueprint plug-in, you may not want the Default value to replace that value. To keep the default value from replacing the plug-in value, set the default to Cohesion.PLACEHOLDER. If set to the placeholder, the plug-in value will not be replaced.

Value Case

Specifies whether CA Configuration Automation ignores text case differences of values.

- Case Sensitive (the default) means CA Configuration Automation identifies and reports all differences.
- Case Insensitive means CA Configuration Automation ignores text case differences.

Translate

Applies a translation to the directive output. If no translation matches, then the original directive result is retained.

Directive values extracted from configuration files and the registry are often cryptic strings or integers belonging to an enumeration. Each value corresponds to a configuration state but it is not clear from the value itself what the interpretation should be. To be clear when displaying these values within the Blueprint, they can be translated to meaningful strings.

Specify a unique translation name in the Translate field. CA Configuration Automation prefixes the name you provide with the string $CCTranslation$__ to make it identifiable as a translation within the database.

Then, define a mapping of values in the CA Configuration Automation Database that references the translation name and triggers the translation of *from values* to *to values*.

The following example defines mapping for a translation called $CCTranslation$__IIS_SERVER_STATE:

```
<row>
  <element_id>23501</element_id>
  <elem_name>$CCTranslation$__IIS_SERVER_STATE</elem_name>
  <elem_dtype>0</elem_dtype>
  <elem_value type="xml"><![CDATA[
    <IIS_SERVER_STATE>
      <Value from="2" to="Running"/>
      <Value from="4" to="Stopped"/>
      <Value from="6" to="Paused"/>
    </IIS_SERVER_STATE>
  ]]></elem_value>
  <elem_desc>Common Enumeration</elem_desc>
  <elem_state>1</elem_state>
  <created_by>1</created_by>
  <creation_time type="date">Jan 14, 2003</creation_time>
</row>
```

Translations do not have to be associated with a particular Component Blueprint. Because they can be referenced by name from any Component Blueprint, ensure you uniquely name each translation to avoid conflict with other translations.

Note: There is no support for the definition of translation tables within the <ACM > Server user interface. Files in CA Configuration Automation data load format (as shown in the example) must be created and loaded to the database using the data loader utility.

Transform

Specifies the XSL transform used to filter the contents when a returned directive value is in the form of XML. The transformed value replaces the original returned value and can be either new XML or any other text generated by the transform.

To execute an XSL transform, specify the location of an XSL file to be applied to the directive value. The transform location can be a URL (file, http, or other) visible from the server or simply a file name that exists within the server CLASSPATH. The transform is retrieved and applied to the directive value to produce a new value for the directive.

Insert

Specifies the value of a directive that needs be inserted within another string to produce the desired result for display or for a subsequent operation.

Any string with $(VALUE) (all capital letters) in it can serve as the insertion location. The $(VALUE) is replaced by the directive value to produce the filtered result. For example, if the initial directive value is 3 and the insertion string is 1.$(VALUE).export, the filtered directive value becomes 1.3.export.

Modifier

Specifies one of the following: Does Host Use Agent or Is Alterpoint Device.

Modifier Parameters

Specifies the parameters applied to the modifier.

Parser Details

Specifies either Structure Class or Parser. Select an option from the corresponding drop-down list.

4. Click the Configuration Data link, then complete the following field:

Database

Specifies the database used by the Blueprint. The drop-down list displays the databases created in Step 22.

Click Save. The database appears in the Configuration Data tree in the left pane.

5. Click Add Query pane, then complete the following fields:

Name

Specifies the directive that executes a query against a database on a managed server, and optionally filters the output to extract a value from the retrieved data.

Common uses for this directive type include:

- Executing a query to extract a column value from a row in the database.
- Executing a stored procedure to make a change on the managed server or to extract a value.
- Executing a query to extract a result set and display it in tabular form in a directive macro step.

The name must be unique in the executables folder of a Blueprint.

Description

Optional text describing the purpose and origin of the executable directive.

This description appears as a tool-tip over the executable directive name in the Blueprint and the parameter name in the tree views.

Query Type

Specifies one of the following query types from drop-down list:

- select—The query is an SQL select statement, data is returned from the execution.
- insert—The query is an SQL insert statement, no data is returned.
- delete—The query is an SQL delete statement, no data is returned.
- update—The query is an SQL update statement, no data is returned.
- others—The query is an SQL alter table or other DDL statement, no data is returned.
- stored procedure—The query string invokes a stored procedure. Data may or may not be returned.

The selected query type directs CA Configuration Automation how to run the query and whether to expect data in return.

Query

Specifies a valid SQL query or stored procedure name (syntax can vary depending on database type). Variable substitution is allowed.

All rows selected by the query or stored procedure are returned. Use the column and regular expression attributes to filter the result set.

Max Rows

Specifies the maximum number of rows retrieved by a query.

Default: 5000

Primary Columns

Specifies the name of the column or an aliased column in the returned result set. The value in the named column is taken as the directive value. If more than one row is returned by the query, the named column in the first row is used.

Column is required for executable directives.

Structure Class

Specifies the structure class (that is, is the hierarchical collection of groups and parameters) used as an overlay to map metadata to the specific set of name-value pairs retrieved from a file by the parser.

Note (Edit Mode only)

Defines a note about the selected element. The number of notes on the element or None is shown in brackets. From the drop-down list, you can view all notes, add a new note, or view, update, or delete an existing note.

If you select Show All Notes, a new page appears showing a list of all notes in table format. You can change the sort order of the table by clicking any of the table headings.

If you select New Note, two additional fields appear:

- Name—Enter a name for the note.
- Text—Enter the note text.

If you select an existing note, the note name appears in the Notes field and the note text is displayed below the name. You can Update, Delete, or Cancel the selected note.

Click Save. The query appears in the Configuration Data tree in the left pane.

6. Click the File Structure Data link, then complete the following fields on the File Structure Class tab:

Name

Specifies the name of the structure class. The name appears in the Blueprint list, and also displays in the tree view unless you specify a Display Name.

Version

Specifies the version of the structure class. The version appears in the Blueprint list, and also displays in the tree view unless you specify a Display Name.

Display Name

Specifies the name you want to appear for the class in the Component Blueprint tree view if it is different than that specified in the Name and Version fields. For example, you could provide text that is more descriptive of the function of the class.

Description

Optional description of the class. This description appears as a tool-tip over the class name in the Blueprint tree view.

Allow Remediation Jobs

Specifies whether to allow changes to the changeable elements of the Blueprint as specified by a remediation job.

Default: Yes

Parser

Specifies the parser to be used to parse files of the defined type. CA Configuration Automation includes a library of predefined lexers and parsers to allow decomposition of most common configuration file formats.

Click Save. The structure class appears in the File Structure Class tree in the left pane.

7. Click the Precedence tab, click Add Group or Add Parameter, then complete the following fields:

Name

Specifies the name of the group.

Description

Describes the purpose of the group.

Available Categories

Specifies a category for the group. Double-click one or more category to move them to the Selected Categories column.

Available Filters

Specifies a filter for the group. Double-click one or more filter to move them to the Selected Filters column.

Weight

Specifies one of the following weights: Low, Medium, or High.

Data Type

Specifies the data type of the element (for example, string, Boolean, or integer).

Valid Values

Specifies the valid values for the parameter or group for a data type range (for example, integer enumeration, integer range, or string enumeration).

Default Value

Fixed string or a string containing one or more variable substitutions. If the directive is run and no Value can be determined, the Default value will be used in place of an undefined or zero length Value.

If the value for a directive can be set by a Component Blueprint plug-in, you may not want the Default value to replace that value. To keep the default value from replacing the plug-in value, set the default to Cohesion.PLACEHOLDER. If set to the placeholder, the plug-in value will not be replaced.

Visibility

Specifies whether the value is shown or hidden as follows:

- Show Value means that the parameter value displays normally. The default is Show Value.
- Hide Value means that the parameter value does not display and instead show as *******. Hide parameters if their values are sensitive (such as passwords) or if the values are binary or too long and cannot be displayed. Hidden parameters are encrypted in the database and are not displayed or viewable in the UI.
- Hide Element lets you hide a parameter. Hide elements if you want the element value available for variable substitution, but do not want to see the element displayed in the service tree view.

Value Case

Specifies whether or not CA Configuration Automation ignores text case differences of values. Case Sensitive (the default) means CA Configuration Automation identifies and reports all differences. Case Insensitive means text case differences are ignored.

Interpret As

Provides a hint about a configuration parameter's string format and its intended use by the associated component. Interpreted parameter values are inspected by CA Configuration Automation using context-sensitive parsers, which allow the extraction of multiple sub-values from within complex parameter strings.

Default: Blank (no interpretation).

Relationship Key

Specifies that CA Configuration Automation uses the Interpret As field to determine and assign relationships if this field is set to Yes. Not all interpretations are capable of defining relationships.

Default: No.

Important! In order to establish relationships, define a proper value in Interpret As field, and set the Relationship field to Yes.

Relationship Type

Specifies the type of relationship for the parameter (for example, manages, hosts, communicates with, or uses).

Interpreted Server Name

Specifies the target database instance or application instance available in any other component parameters. Use variable substitution methods to define the parameter.

Interpreted Source Instance

Specifies the source database instance or application instance.

Interpreted Target Instance

Specifies the target database instance or application instance.

Interpreted Application Name

Specifies the target application information available in any other parameters. Use variable substitution method to define the parameter.

Expected Value

Specifies the expected value of the parameter or group.

Value

Specifies a unique value of the group in the file.

Qualifier Child (Add Group only)

Specifies the parameter used as a qualifier under the group.

8. Click Next.

   The Macros page appears.

9. Enter the appropriate information in the following fields, then click Next:

   Name

   Specifies the name of the macro (a macro is a sequence of operations, each returning a value and a status).

   Description

   Describes the purpose of the macro.

Folder

Specifies the folder where the macro is located.

Diagnostic

Specifies whether or not the macro is used for diagnostic purposes.

Default: No

Read Only

Specifies whether or not the macro can alter the target system.

Default: No (the target system cannot be altered).

To execute a Read-Only Macro, the user must have Server View or Service View permissions.

The Component Grouping Options page appears. It contains nesting options that let you emphasize the relationship between components in a service by embedding one component within another when displayed.

When, for example, a software component uses and depends on several subordinate components, and those components are installed within the primary component's file system root, nesting can be used to enforce the parent-child relationship between them.

Oracle databases, for example, normally install a Java Runtime Engine and an Apache Web server within their installation directory. The relationship between the utility components and the Oracle database is expressed by nesting the JRE and Apache within the Oracle component.

10. Enter the appropriate information in the following fields, then click Finish:

Allow this blueprint to contain other components

Specifies whether this Blueprint can contain other components. If this check box is blank, the Blueprint will not contain other components.

Allow this blueprint to nest in other components

Specifies whether this Blueprint can be nested in other components. If this check box is blank, the Blueprint will not nest in other components or have other components nested within it.

Allow this blueprint to contain only named components

Specifies whether this Blueprint can contain only named components.

This blueprint will not use nesting

Specifies whether this Blueprint can use nesting.

Refresh Order

Defines the position that this component will refresh relative to all others on a given server. During Refresh, the components on each server are refreshed sequentially (components on other servers are refreshed in parallel).

Refresh order is important when components depend on each other for variable substitution. For example, if one component uses the value of a parameter from another component for variable substitution, it is important that this other component be refreshed first. If Refresh Order is not specified in this case, the dependent component can pick up a stale value that has not yet been updated by the current refresh.

The following seven levels, from Initialization to Cleanup can be selected (within a given level, no particular order is guaranteed. If Don't Care is selected, the component is treated as if it has the Middle ordinal):

- Don't Care. No preference, but is treated as Middle if other components are present.
- Initialization. The first group to be refreshed.
- First
- Early
- Middle
- Late
- Last
- Cleanup. The last group to be refreshed.

Modifier

Specifies one of the following:

- Device Authority
- IIS Nesting
- Replace All
- Sun Application Server Nesting
- TIBCO
- TIBCO Nesting

Modifier Parameters

Specifies the parameters applied to the modifier.

Nearest Neighbor

Specifies that the component with the closest file system root (in terms of depth) be selected if a component can nest in more than one other component.

Nest By

Defines the file system relationships that determines if this component nests within other components. Select one of the following values from the Nest By drop-down list (if the This Blueprint Will Not Use Nesting check box is selected, this value is not required):

- Child—Specifies that this component should nest in any other component whose file system root is above it in the file system directory tree.
- Child or Sibling—Specifies that this component should nest in any component whose file system root is equal to or above it in the file system directory tree.
- Direct Child—Specifies that this component can nest only in components whose file system root is one level above its own in the file system directory tree.
- Sibling—Specifies that this component should nest in any other component with the same file system root.

Nest Only In

Specifies the Blueprint that uses the nesting defined on this page. Double-click a Blueprint in the Available Component Blueprints column to move it to the Selected Component Blueprints column.

Path from Root

Specifies the component only nests in another component whose file system root is located above it, with the exact relative path between them. This option can be used to select one out of several potential nesting matches or to restrict nesting to an exact location within the file system directory tree.

Depth from Root

Specifies the component only nests in another component whose file system root is located above it, with the exact depth between them. This option can be used to select one out of several potential nesting matches or to restrict nesting to an exact depth within the file system directory tree.

The Blueprint is created and appears in the Blueprint table.

Chapter 4: Blueprint Element Reference

The topics in this section provide a reference to the various blueprint elements that can be used to discover and manage software components.

This section contains the following topics:

Category Descriptions

The Category field lets you organize Component Blueprint elements.

Category	Description
Administration	Administrative settings that have to do with the general availability and management of the component. Examples of administrative settings would be how to back up, when to flush a cache, or how many times to retry something.
Configuration	Configuration settings for the component, except for those that are better described by Administration, Log and Debug, Network, or Performance. Examples of configuration settings would be a component alias name or default web page.
Documentation	Elements that document the component behavior or act as a guide to users, for example manuals, readme files, FAQs, or online help pages.
Log And Debug	Elements that have to do with setting up such things as log locations, log levels, debug output, or diagnostic variable types.
Network	Elements that represent or indicate a network-related setting for the component. An example would be the port settings for SNMP. If an element can be categorized as both Network and Security (for example, enabling LDAP Authentication), use Security as the category.
Other	CA internal use only.

Performance	Settings that are known to seriously impact performance and are generally a specific subset of configuration parameters. Examples of performance settings would be number of threads or number of concurrent users.
Product Info	General (and usually static) information about the product. Examples of static component information would be licensing, installation location, vendor, or module name.
Resources	Component resources. Examples of component resources would be storage, memory and cache allocation or size, or CPU. Note that this static resource category is different than the Transient category, which relates more to real-time information.
Security	Elements that represent security-related settings and are generally a specific subset of configuration parameters. Examples of security settings would be authentication types, enabling authentication, encryption settings, directory browsing, SSL, or HTTPS.
Transient	Elements that change with some regularity. Examples of transient elements would be server states (for example, up, down, running, or stopped), current number of connected clients, current number of threads, or current disk utilization.
Versioning and Patches	Elements that indicate the version or patch levels of the product.

Filter Descriptions

The Filter field lets you mark Component Blueprint elements for exclusion from key CA Configuration Automation operations and results like Change Detection and Rule Compliance.

Filter	**Description**
Component Specific	Identifies an element that is specific to a single component instance, for example, installation root and Service Server Name. The purpose is to identify and exclude certain component-specific elements, which are already known to be different, from Change Detection operations and results.
Service Specific	Identifies an element that is specific to a service, for example server names and installation roots. The purpose is to identify and exclude certain service-specific elements, which are already known to be different, from Change Detection operations and results.

Server Specific	Identifies an element that is specific to a single server, for example Server name and IP address. The purpose is to identify and exclude certain server-specific elements, which are already known to be different, from Change Detection operations and results.
Never Run Change Detection	Identifies an element that should be permanently excluded from all types of Change Detection operations and results. A temporary directory, log files in the managed folder, or anything that is known to be transient are examples of elements that you want to identify and always exclude from Change Detection operations and results.
Never Run Rule Compliance	Identifies an element that should be permanently excluded from all types of Rule Compliance operations and results due to their inconsequential or variable nature. A temporary directory, a known old configuration file, a template, or an example file are examples of elements you want to identify and always exclude from Rule Compliance operations and results.
Time Variant	Identifies an element that is known to change over time, but not necessarily across servers or across services, for example, log files, process start times, and registry event counters. The purpose is to identify and exclude parameters that are time variant, which are already known to be different, from Change Detection operations and results.

POSIX 1003.2-1992 Pattern Matching

POSIX pattern matching expressions are descriptions that help you find files and directories when the exact file or directory name is not known. CA Configuration Automation uses POSIX pattern matching expressions to search for files and directories during discovery and during refresh operations.

Syntax of POSIX Pattern Matching Expressions

File Name Matching Expression	Description
Characters	
unicodeChar	Matches any identical Unicode character.
?	Matches any single character.

*	Matches any string, including the null string (zero length).
Character Classes	
[abc]	Matches any character listed within the square brackets (simple character class).
[a-zA-Z]	Matches any range of characters listed within the square brackets (character class with ranges).
[^abc]	Matches any range of characters *except* those listed within the square brackets (negated character class).
Standard POSIX Character Classes	
[:alnum:]	Matches alphanumeric characters
[:alpha:]	Matches alphabetic characters
[:blank:]	Matches space and tab characters
[:cntrl:]	Matches control characters
[:digit:]	Matches numeric characters
[:graph:]	Matches characters that are printable and visible. (A space is printable but not visible, while an a is both.)
[:lower:]	Matches lowercase alphabetic characters
[:print:]	Matches printable characters (characters that are not control characters)
[:punct:]	Matches punctuation marks (characters that are not letters, digits, control characters, or space characters)
[:space:]	Matches characters that create space, such as tab, space, and formfeed
[:upper:]	Matches uppercase alphabetic characters
[:xdigit:]	Matches characters that are hexadecimal digits

Variable Substitution

Variable substitution allows the value of any element managed by CA Configuration Automation to be used as a parameter within a Component Blueprint directive. CA Configuration Automation defines an expression syntax to identify elements in the blueprint. Once identified, the element's value is extracted and used in place of the expression when the directive is executed. Substitution can be applied to any attribute of a directive. These include values, default values, paths, file names, parameters, environment variables, regular expressions, queries, and column names.

Managed elements whose values are addressable by variable substitution syntax include:

- Discovery parameters
- Registry variables
- Configuration values (file, database, or executable)
- File and directory attributes
- Managed data elements (schema metadata)
- Service and component attributes

Expression Types

Variable substitution expressions have the following forms:

Parameter Substitution

Provides access to the values of parameters that are defined in the current component.

Object Substitution

Lets you address the value of any element in any service.

Global Variable Substitution

Lets you address the values from the CA Configuration Automation global variable repository.

The following sections describe these substitution types.

Parameter Substitution

Parameter substitution expressions have the form:

`$(`*`VariableName`*`)`

VariableName

Identifies a discovery parameter that is defined in the current component. The name is case-sensitive, so it must match the parameter name exactly. You can embed the parameter expressions in string literals or you can use them standalone. You can define multiple substitutions in the same expression, and you define them recursively. For example, the substitution value can be a string in the parameter expression. If so, the product evaluates the substitution value recursively.

Example:

If you define the following Discovery parameters:

```
User=info
Domain=ca.com
v1=$(User)
v2=$(Domain)
```

the following parameter substitution expression:

```
$(User)@$(Domain) [$(v1) at $($(v2))]
```

returns the following result after evaluation:

```
info@ca.com [info at ca.com]
```

Object Substitution

Object substitution expressions define the path to an object in the CA Configuration Automation managed element tree. When an object is identified, the product returns the object value as the result of the expression. Alternatively, the product can return an attribute of the object, as the example in the Component-Scoped Object Substitution Expressions shows. If no object matches the expression, the product returns a null value.

You can define object substitution expressions in the scope of a service, in the current component, or globally.

Service-Scoped Object Substitution Expressions

A service-scoped object substitution expression must specify a component that can exist in the service. Service-scoped object expressions have the form:

```
${Component[ComponentName,ElementType[ElementName or Identifier,
...]]}
```

Braces { }

Distinguish the object syntax from the parameter expression syntax.

ComponentName

Defines either the name of a single component or a list of components that is delimited with the | character. The delimited list variant lets the object expression resolve a value when the service database contains multiple components.

Examples:

To use a delimited list variant where the component could be either SQL Server or Oracle:

```
${Component[Microsoft SQL Server|Oracle 8i
Server,Parameter[DatabaseUser]]}
```

To access the root parameter of a component:

```
${Component[CCA Server,Parameter[Root]]}
```

You can also select components by the Component Blueprint category:

```
${ComponentCategory[Relational Databases,Parameter[DatabaseUser]]}
```

The Component Blueprints page lists the valid category names:

- Application Platforms
- CA Software
- Clustering
- Compliance
- Custom Components
- Directory Servers
- Enterprise Applications
- Imported Components
- IT Management Systems
- Messaging Systems
- Network Devices
- Operating Systems
- Operating Systems - Limited
- Relational Databases
- Server Components
- Storage Managers
- Utilities
- Virtualization
- Web Servers

Component-Scoped Object Substitution Expressions

Component-scoped object expressions have the form:

```
${ElementType[ElementName or Identifier, ...]]}
```

Example:

```
${FileSet[$(Root),Directory[admin/logs,File[filter.log,Attribute[s
ize]]]]}
```

Globally Scoped Object Substitution Expressions

Globally scoped object expressions can access information from any service in a single CCA Database. Globally scoped object expressions have the form:

```
${Service[ServiceBlueprintName(ServiceName),Component[ … ]]}
```

Example:

To get the CA Configuration Automation mail from a configuration parameter from a service other than CA Configuration Automation:

```
${Service[CCA(MyCCA),Component[CCA Server,Configuration
[*,Files[*,Directory[lib,File[cca.properties,FileStructure[*,NVFil
e
[com.ca.mail.from]]]]]]]}
```

Elements and Attributes that are Available for Object Substitution Expressions

The following tree enumerates:

- The specific element types
- The relationships of specific element types in the tree
- The object values
- The available attributes (in parentheses)

You can use the strings in this example in an object substitution expression to build a path to an object in the tree.

```
Component [name or id]
   (module_id, mod_name, mod_desc, mod_version, platform_id
   mod_instance_type, mod_instance_of, release_version, mod_state,
   created_by, creation_time, server_id, server_name, domain_name,
   ip_address, mac_address, server_state,
   cc_agent_yn,cc_agent_port,
   cc_agent_protocol, os_type, os_version, processor, platform_name)
Parameter [parameter name]
Files [$(Root)]
   Directory [directory name or path (a/b/c)]
      (name, mtime, ctime, owner, perm, bytes, depth, files,
      directories)
   Directory ...
   File
   File [file name]
      (name, mtime, size, owner, perm, prodver, filever, ctime)
   Registry [*]
      RegKey [keyname or path (a\b\c)]
      (name, value)
   RegKey ...
   RegValue [name]
   (name, value)
   Configuration [*]
      Files [*]
       File [name]
        FileStructure
         GroupFileBlock [name]
         GroupFileBlock [name(value)] where value is the group
      block's
                  value, name qualifier, or name qualifier child
      value.
         GroupFileBlock ...
          NVFileBlock [name]
          NVFileBlock [name]
                  (description, view, weight, password, folder)
   Database [name]
      ResultSet [name]
```

```
        (name, type, query, queryType, description)
    DataRow [name]
    DataCell [name]
        (name, value)
DatabaseKey [name]
        (name, description, key, keyValues, column)
ExecutablesFileSystem [*]
    File [name]
    FileStructure
    GroupFileBlock [name] or GroupFileBlock [name(value)] where
    value
        is the group block's value, name qualifier, or name
    qualifier
        child value.
    GroupFileBlock ...
    NVFileBlock [name]
        (description, view, weight, password, folder)
Database [database name]
    DataBaseAccessSpec
        (server, user, password, driver, databaseName,
    databaseContext, env)
    Table [table name]
        (name, description, rowcount)
    Column [column name](name, description, length, nullable,
    default, ordinal, precision)
    Index [index name]
        (name, sort, unique, description)
    Column [column name]
        (name, description, length, nullable, default, ordinal,
    precision)
```

Global Variable Substitution

Global variable substitution expressions have the form:

$(*GlobalVariableName*)

GlobalVariableName

Defines a valid path in the CA Configuration Automation global variable repository. The name is not case-sensitive. You can embed the global variable expressions in strings or you can use them standalone. You can define multiple substitutions in the same expression, and you can define them recursively. For example, if the substitution value is a string in the parameter expression, the application evaluates the substitution value recursively.

Example:

For a global variable repository with the following structure:

```
Global Variables
    Site
        Phoenix
          Main: x4000
          Fire: x4911
        Tucson
          Main: x5000
          Fire: x5911
```

the following global variable substitution expression:

```
$(/Site/Tucson/Main)
```

returns the following result after evaluation:

```
x5000
```

Interpret As Descriptions

Interpret As provides a hint to CA Configuration Automation about a configuration parameter string format and how it is intended for use by the associated component. The application uses context-sensitive parsers to inspect interpreted parameter values, which lets the application extract multiple subvalues from complex parameter strings.

For example, if CA Configuration Automation can interpret and extract the following value as a JDBC URL, it can extract the database type, server, port, and database name:

```
jdbc:oracle:thin:@dbserver:1521:MYDBNAME
```

In addition to enabling context-sensitive parsing, interpretation also lets the application derive relationships. Using the extracted server in the example above, the application can establish a relationship between the current server and the server dbserver. To establish a relationship, use the Relationship Key.

The application allows only one interpretation for each value, and many values have no interpretation (leave such values uninterpreted). If more than one interpretation applies (for example, File Name and File Name or Path), use the one that most accurately describes the field. For example, if a field is defined as a file name (with no path), select File Name. If the field can be a file name, path, or partial path, select File Name or Path. The application includes the following Interpret As selections:

Database Name

The value is the name of a database in a database server.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Database Table

The value is the name of a database table in a database. The database table can contain a schema prefix.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Date

The value is a date in any format.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Date And Time

The value is a date that is combined with the time of day.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Description

The application interprets the value as descriptive text.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Directory Name

The value is only a directory with no path.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Directory Name or Path

The value is a directory name, path, or partial path.

Because the application can derive the Directory Reference relationships from this interpretation, you can set Relationship Key to Yes.

Email Address

The value is the destination for an email message. The interpreter looks for one or more email addresses in the value string.

File Name

The value is only a filename with no path.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

File Name or Path

The value is a file name, path, or partial path. Many named values allow any of these interpretations.

Because the application can derive the File Reference relationships from this interpretation, you can set Relationship Key to Yes.

Server Name or IP Address

The value is (or contains) an IP address or server name. Use this interpretation only when no port number is defined in the value. If the value contains a port number, use Server Name and Port.

The application can recognize server names and IP addresses that are embedded in larger strings.

Because the application can derive the Server Reference relationships from this interpretation, you can set Relationship Key to Yes.

Define a Server Reference relationship as a relationship key only if the referenced server is considered a dependency of the current server.

Server Name and Port

The value is (or contains) a server name or IP address and port number. A colon (:) must separate the server and port number.

The application can recognize server names, IP addresses, and port numbers that are embedded in larger strings.

Because the application can derive the Server Reference relationships from this interpretation, you can set Relationship Key to Yes.

Specify a Server Reference relationship as a relationship key only if the referenced server is considered a dependency of the current server.

Java Class Name

The value is a Java class name. It can be a class name, a package name, or a fully qualified class name with a package prefix.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

JDBC URL

The value defines a JDBC URL. The table shows the supported formats.

Because the application can derive the Server Reference relationships from this interpretation, you can set Relationship Key to Yes.

JDBC URLs almost always define an important relationship. You should typically identify them as Relationship Keys.

LDAP Path

The value defines a path to an LDAP subtree.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

LDAP Entry

The value is the name of an LDAP directory entry or the full path to an entry.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Network Domain

The value is a network domain (not including the server name). For example:

`ca.com.`

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Network Protocol

The value defines an IP protocol, such as TCP, UDP, FTP, SNMP, or SMTP.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Password

The value is a password.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Registry Key Name

The value is only a registry key name with no path.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Registry Key Path

The value is the full path (starting with \) to a registry key.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Registry Value Name

The value is only a registry value name with no path.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Registry Value Path

The value is the full path (starting with a \) to a registry value.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

SNMP Community String

The value specifies an SNMP community string. For example:

```
public
```

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

SNMP OID

The value specifies an SNMP object ID. For example:

```
1.3.6.1.4.1.18071.1.1.1
```

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

TCP Port Number

The value is a TCP port number (not UDP or unspecified).

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Time Interval

The value is an interval of time.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Time Of Day

The value is the time of day.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

UDP Port Number

The value is a UDP port number (not TCP or unspecified).

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

URL

The value specifies a URL, including the following protocols: file, http, https, ftp, jrmi, jmx:rmi, iiop, gopher, news, telnet, mailto, jnp, t3, and ldap.

The interpreter decomposes the URL and makes parts of it available through custom methods.

Because the application can derive the Server Reference relationships from this interpretation, you can set Relationship Key to Yes.

Define a URL relationship as a relationship key only if:

- The URL contains a server name
- The named server always defines a dependency between the current server and the named server.

User Group

The value is a user group.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

User Name

The value is a user name.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Version String

The value is any string that can be interpreted as a version.

Because no relationships are derived from this interpretation, always set the Relationship Key to **No**.

Web Service URL

The value specifies a URL that identifies a web service that a component uses.

Because the application can derive the Server Reference relationships from this interpretation, you can set Relationship Key to Yes.

The JDBC URL relationship interpretation supports the following formats:

Database Name	URL Pattern
SQL Server2005 SQL Server 2008	jdbc:sqlserver://<host>:<port>;databasename=<database>; SendStringParametersAsUnicode=false

Database Name	URL Pattern
Oracle 9, 10, and 11	■ jdbc:oracle:thin:@$(host):$(port):$(database) ■ jdbc:oracle:thin:@<host>:<port>:<database> ■ jdbc:oracle:thin:@<host>:<port>/<database> ■ jdbc:oracle:thin:@((DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=<host1>)(PORT=<port1>)(ADDRESS=(PROTOCOL=TCP)(HOST=<host2>)(PORT=<port2>))(FAILOVER=ON)(LOAD_BALANCE=OFF)(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=<database>)) ■ jdbc:oracle:thin:@((DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=<host>)(PORT=<port>)))(CONNECT_DATA=(SERVICE_NAME=<database>))) ■ jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=<host>)(PORT=<port>)))(CONNECT_DATA=(SERVICE NAME=<database>))) ■ jdbc:bea:oracle://<host>:<port>
Informix	jdbc:informix-sqli://<host>:<port>/<database>:informixserver=<serverName
DB2	jdbc:db2://<host>:<port>/<database>
Sybase 11 and 15	jdbc:sybase:Tds:<host>:<port>/<database>
MYSQL	■ jdbc:mysql://<host>:<port>/<database> ■ jdbc:mysql://<host>:<port>
Postgres	jdbc:postgresql://<host>:<port>/<database>
HSQLDB	jdbc:hsqldb:hsql://<host>:<port>
ODBC	jdbc:odbc:<database>
Cloudscape	jdbc:cloudscape:<database>
Java DB (Derby)	■ jdbc:derby://<host>:<port>/<database> ■ jdbc:derby://<host>:<port>/<database>;create=true
Ingres	jdbc:ingres://<host>:<port>/<database>
Pointbase	jdbc:pointbase:server://<host>:<port>/<database>
Generic	jdbc:<xyz>:server://<host>:<port>/<database>

Regular Expressions

Regular expressions are pattern descriptions that enable sophisticated matching of strings. CA Configuration Automation uses regular expressions to:

- Search files for matching strings
- Verify that the string conforms to certain patterns, such as email addresses
- Extract string values from large blocks of text

If you need more information about the concepts behind regular expressions, there are many sources on the Web. For example, the Google directory about computer programming languages includes a helpful section about regular expressions: http://directory.google.com/Top/Computers/Programming/Languages/Regular_Expressions/FAQs,_Help,_and_Tutorials.

Regular Expression Syntax

The following table shows the supported syntax for regular expressions in CA Configuration Automation.

Regular Expression	Description
Characters	
unicodeChar	Matches any identical Unicode character.
\ (backslash)	Used to quote a meta-character or to process a special character as normal or literal text. For example, * makes the asterisk a normal text character, instead of a wildcard character.
\\	Matches a single \ character.
\0nnn	Matches a specified octal character.
\xhh	Matches a specified 8-bit hexadecimal character.
\uhhhh	Matches a specified 16-bit hexadecimal character.
\t	Matches an ASCII tab character.
\n	Matches an ASCII newline character.
\r	Matches an ASCII return character.
\f	Matches an ASCII form feed character.
Character Classes	
[abc]	Matches any character between the square brackets (simple character class).

[a-zA-Z]	Matches any range of characters between the square brackets (character class with ranges).
[^abc]	Matches any range of characters *except* those between the square brackets (negated character class).
Standard POSIX Character Classes	
[:alnum:]	Matches alphanumeric characters.
[:alpha:]	Matches alphabetic characters.
[:blank:]	Matches space and tab characters.
[:cntrl:]	Matches control characters.
[:digit:]	Matches numeric characters.
[:graph:]	Matches characters that are printable and visible. (A space is printable but not visible, but an *a* is both.)
[:lower:]	Matches lowercase alphabetic characters.
[:print:]	Matches printable characters (characters that are not control characters).
[:punct:]	Matches punctuation marks (characters that are not letters, digits, control characters, or space characters).
[:space:]	Matches characters that create space (for example, tab, space, and formfeed characters).
[:upper:]	Matches uppercase alphabetic characters.
[:xdigit:]	Matches characters that are hexadecimal digits.
Non-Standard POSIX-Style Character Classes	
[:javastart:]	Matches the start of a Java identifier.
[:javapart:]	Matches part of a Java identifier.
Predefined Classes	
. (period)	Matches any character other than newline.
\w	Matches a “word” character (alphanumeric plus “_”).
\W	Matches a non-word character.
\s	Matches a whitespace character.
\S	Matches a non-whitespace character.
\d	Matches a decimal digit.
\D	Matches a non-digit character.

Boundary Matches	
^ (caret)	Matches only at the beginning of a string.
$ (dollar sign)	Matches only at the end of a string.
\b	Matches any character that is at the beginning or end of a word boundary.
\B	Matches any character that is not at the beginning or end of a word boundary.
Greedy Closures (Also known as *quantifiers*. For more information, see the Note that follows this table.)	
A*	Matches A zero or more times.
A+	Matches A one or more times.
A?	Matches A one or zero times.
A{*n*}	Matches A exactly *n* times.
A{*n*,}	Matches A at least *n* times.
A{*n*,*m*}	Matches A at least *n* but not more than *m* times.
Reluctant Closures (Also known as *quantifiers*. For more information, see the Note that follows this table.)	
A*?	Matches A zero or more times.
A+?	Matches A one or more times.
A??	Matches A zero or one times.
Logical Operators	
AB	Matches A followed by B.
A|B	Matches either A or B.
(A)	Parentheses are used to group subexpressions.
Back References (Reaches back to what a preceding grouping operator matched and uses it again to match something.)	
\1	Back reference to first parenthesized subexpression match.
\2	Back reference to second parenthesized subexpression match.
\3	Back reference to third parenthesized subexpression match.
\4	Back reference to fourth parenthesized subexpression match.
\5	Back reference to fifth parenthesized subexpression match.
\6	Back reference to sixth parenthesized subexpression match.

\7	Back reference to seventh parenthesized subexpression match.
\8	Back reference to eighth parenthesized subexpression match.
\9	Back reference to ninth parenthesized subexpression match.

Note: All closure operators (+, *, ?, {*m*,*n*}) are "greedy" by default. That is, they match as many elements of the string as possible without causing the overall match to fail. To use a "reluctant" (non-greedy) closure, follow it with a ? (question mark).

Java Plug-ins Supplied with CA Configuration Automation

Optionally, Java plug-ins implementing the com.ca.catalyst.object.CCICatalystPlugin interface can filter directive values. You can develop the plug-ins and then add them to the CLASSPATH or use one of the following plug-ins that are supplied with CA Configuration Automation:

com.ca.catalyst.plugin.CCParameterRuleFilter(*pattern*)

Formats the Version parameter. This filter only recognizes and alters directives named Version.

For example, if the Version value initially extracted from a file is 530, specifying the CCParameterRuleFilter(#.#.#) plug-in translates the Version to 5.3.0. Specifying the CCParameterRuleFilter(#.##) translates the Version to 5.30.

CCMatch(*regex*)

CCMatchAnywhere(*regex*)

Returns "true" or "false" values by matching the specified regular expression with the directive value.

- If the regular expression exactly matches the entire value, CCMatch() returns "true."
- If the regular expression matches anywhere in the value, CCMatchAnywhere() returns "true."

The application interprets the regular expressions with DOTALL and MULTILINE mode enabled. DOTALL mode implies that the regular expression character '.' matches all characters, including end of line characters. MULTILINE mode implies that the regular expression characters '^' and '$' delimit lines, instead of the entire value beginning to end.

CCReplaceAll("*regex*","*replacement*")

CCReplaceFirst("*regex*","*replacement*")

Replaces the portions of the value that match the regular expression.

Surround the regular expression and the replacement with quotation marks, and separate them with a comma. To replace the regex value with a carriage return, use the \n special character in the replacement string. For example, CCReplaceAll(" ","\n") replaces all spaces with carriage returns.

CCToUpper

CCToLower

Converts the directive value to upper or lower case.

CCTrim

Removes leading and trailing spaces from the value.

CCExpression

Runs the specified expressions that contain the directive value.

Expressions are written in ECMA-script (JavaScript) and can contain any syntax that is valid in Version 2 of that language. Include the value of the current parameter as $(*VALUE*) in the expression. The parameter must have a value or the application does not call the plug-in.

The application allows variable substitution in the expression. For example:

CCExpression($(*VALUE*)*50)

Multiplies the value by 50.

CCExpression('$(*VALUE*)' == 'XXX')

Returns "true" or "false".

CCExpression(Math.sqrt($(*VALUE*)))

Takes the square root of the value.

CCExpression(function add(a,b){return a+b;};add($(*VALUE*),$(Other));)

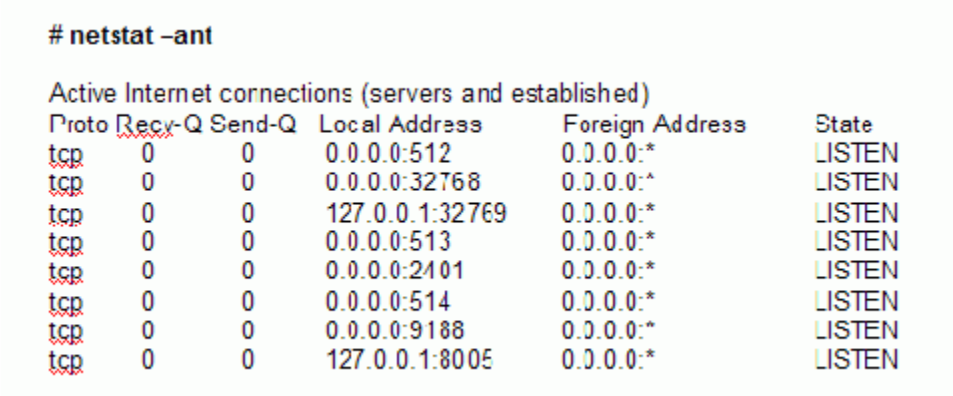
Defines and calls functions.

Understanding and Using the Tabular Data Parser

Tabular data is any text that is formatted in rows and columns. Tabular data can also include embedded comments and one or more heading rows.

Examples of Tabular Data

- Output of the netstat command:



 # netstat –ant

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:512	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:32768	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:32769	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:513	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:2401	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:514	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:9188	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:8005	0.0.0.0:*	LISTEN

- Tab-separated output from an Excel spreadsheet:

Host	Address	Type	Owner
Bertha	192.168.123.12	Linux	Jerome
Factotum	192.168.123.33	Windows 2008	Bukowski
Terrapin	192.168.124.13	AIX	Hunter

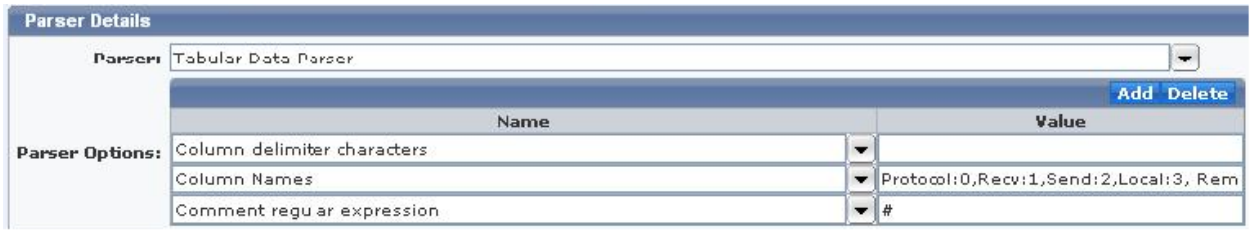
CA Configuration Automation provides a Tabular Data Parser that interprets and parses any form of tabular data within configuration files or executables. In addition, you can specify parser options that control the layout of rows and columns within the tabular data set, assign names to columns, eliminate header and comment text, as well as organize the data hierarchically.

You can also use the Tabular Data Parser and specify parser options at the structure class-level, however file- and executable-level assignments take precedence over any assignments made at the structure class level.

Accessing and Using the Tabular Data Parser

To use the Tabular Data Parser, follow these steps:

1. In the class, file, or executable attribute sheet, select Tabular Data Parser from the Parser drop-down list.
2. Define the details of the tabular layout in the Parser Option(s). The following illustration displays the parser details:



Parser Options are a set of attributes that are listed in a dropdown. Use the Add and Delete to add or delete an Option. Click the Name and Value option to edit the options.

For example, the Parser Options that are listed in the illustration are as follows:

- Column delimiter method
- Column Names
- Comment regular expression

3. Click save.

The Parser Option(s) field is updated.

Parser Options

You can set the following options for the Tabular Data Parser in the Parser Option(s) field.

Note: Enclose all values that you supply for an option in parentheses.

Column delimiter characters=

Defines one or more column delimiters.

If you do not define this option, the application defaults to the tab character. You can also specify more than one column delimiter.

For example, to specify the colon, slash, and comma as potential delimiters:

```
Column delimiter characters=:/,
```

For example, to specify the space, tab, or both as delimiters:

Space only

```
Column delimiter characters=" "
```

Tab only

```
Column delimiter characters="    "
```

Tab and space

```
Column delimiter characters="     "
```

Note: The quotation marks (" ") are only used for illustration. In actual practice, type only the space or tab character without enclosing quotation marks.

Column delimiter method=

Defines how to process consecutive delimiters. You can set this option to "one" or "all."

If you do not define this option, the application defaults to "all" and it processes consecutive delimiters as a single delimiter. For example, if you specify:

```
column delimiter characters=,
column delimiter method=all
```

for the following data:

```
ftp,tcp,udp,,,,xyz
```

the parser returns four columns: ftp, tcp, udp, xyz.

Set the value to "one" to process multiple consecutive spaces as one delimiter or to include data columns even if they have no defined values. For example, if you specify:

```
column delimiter characters=:
column delimiter method=one
```

for the following data:

```
root::0:XDCGBH!:
```

the parser returns the following columns:

```
root, "", 0, XDCGBH!, ""
```

Note: If you specified "column delimiter method=all in the previous example, the parser would return only the following columns:

```
root, 0, XDCGBH!
```

Header count=

Defines the number of lines to ignore at the beginning of the data set. For example, you can use:

```
Header count=2
```

to eliminate the header information from the tabular data results of netstat command.

You can only eliminate complete lines with the Header count= option. The parsed file does not include lines that you remove in the CA Configuration Automation UI.

Comment regular expression=

Defines a regular expression that identifies comments in the data set. For example, if you specify:

```
Comment regular expression=#.*
```

the parser interprets and ignores as comments patterns that start with # (including all other characters to the end of a line). For example:

```
#
# These three lines are removed
#
```

The parser also uses this option to interpret and ignore partial lines. For example:

```
some data   # this comment is also removed
```

The parsed file does not display lines and partial lines that you remove in the CA Configuration Automation UI.

Column Names=

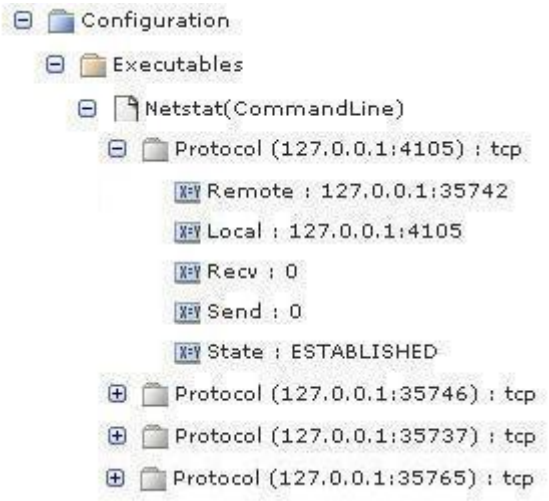
Defines the assignment of names to individual columns of data. The format of the field is a comma-delimited list of names and column index numbers. Column indexing starts at zero. For example:

"Column Names"=Protocol:0,Recv:1,Send:2,Local:3,Remote:4,State:5

The parsed file does not display columns that are that you exclude from the Column Names= option in the CA Configuration Automation UI.

An important aspect of parsing tabular data to the standard CA Configuration Automation internal data format is the structuring of data into Structure Class groups. Groups allow you to assign each row of data a unique qualifier, nest them, and display them hierarchically on the user interface. The Tabular Data Parser uses the first name:index pair that is defined in the Column Names= option to name the group that contains the data rows (the *group pivot*).

With the Column Names= option in the previous example and the netstat command output as input, the application would display the data as follows:

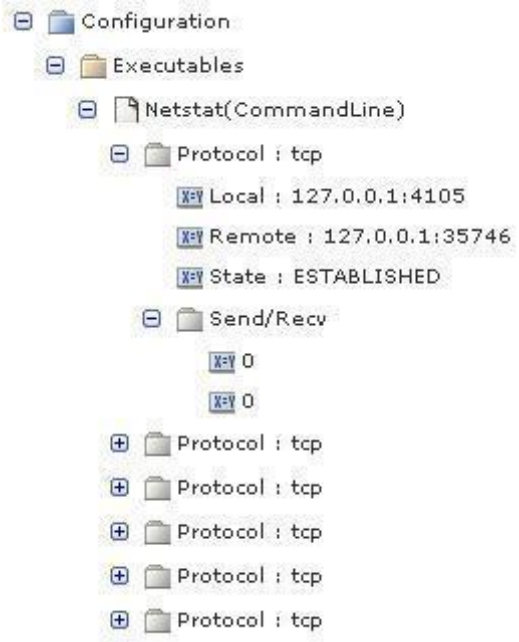


Note: The Structure Class used to interpret the parsed data in this example specifies Local as the qualifier for Protocol.

You can also group multiple columns to form subgroups under the top-level group pivot. For example, to nest the Recv and Send columns by one level in the hierarchy, apply the group modifier as follows:

"Column Names"=Protocol:0,Send/Recv(group):1-2,Local:3,Remote:4,State:5

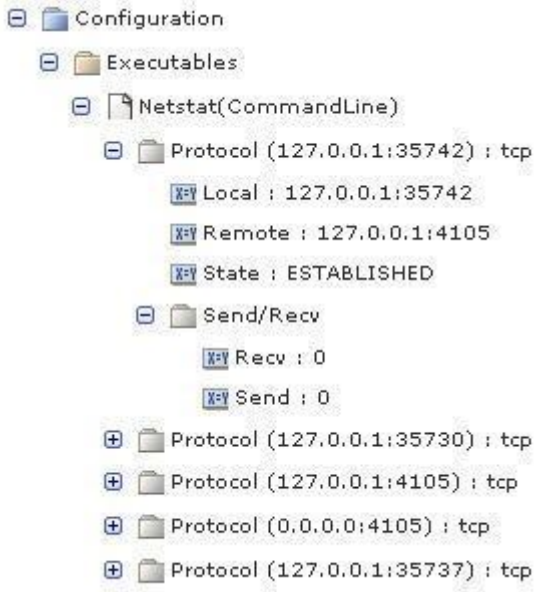
The data would be displayed as:



Note: In this example, the nested values are displayed under the named group and do not have names. The lack of names can limit the ability to write Structure Classes that accurately qualify the parent group. To define names for the nested columns, specify name:index pairs for the columns that are nested after the group modifier. For example:

```
"Column Names"=
Protocol:0,Send/Recv(group):1-2,Recv:1,Send:2,Local:3,Remote:4,
State:5
```

The application displays the data as:



In the name:index pair specification, you can define groups as ranges of columns, lists of individual columns, or a combination. Valid group column formats include:

5-

Column 5 and all successive columns

3-5|7-

Columns 3 through 5 and 7 and all successive columns

3-5|7|9-11

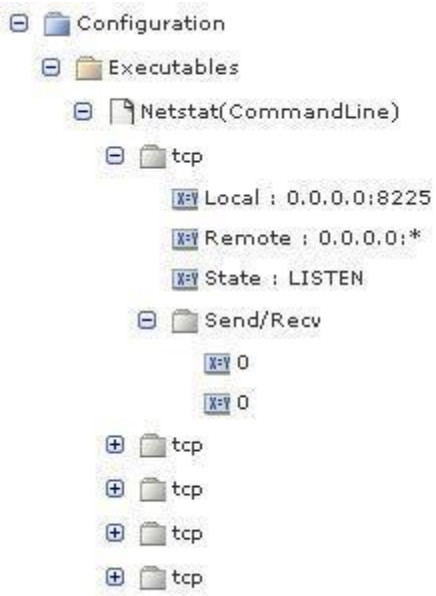
Columns 3 through 5, 7, and 9 through 11

Note: You cannot specify the group option for the first name:index pair because it is the group pivot on which the application builds the hierarchy.

To substitute the value of a column for the name that a name:index pair specifies, use the valueasname modifier. You can specify the valueasname modifier on the group pivot (the first name:index pair). This is a convenient way to display tabular data because one column in a table is commonly a unique key. For example, if you applied the valueasname modifier as follows:

```
"Column
Names"=Protocol(valueasname):0,Send/Recv(group):1-2,Local:3,Rem
ote:4,State:5
```

The application displays the data:



Line continuation regular expression=

Defines a regular expression to identify the line continuation syntax.

For example, use the line continuation character \ to continue a single row of data across multiple lines in the following file:

Name	Phone	Email	Address
Fred	9000	Fred@acme.com	12Jones Cl.
Ame	3002	Ame@acme.com	33535 Eucalyptus Terrace
Mary	7331	Mary@acme.com	31 Main Street

To parse the data correctly, use the following Line continuation regular expression= option:

```
Line continuation regular expression= \\$
```

Note: The first \ escapes the second \ to form a valid regular expression.

Index