CA Configuration Automation®

Blueprint Developer Guide

r12.8 SP01

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 Management in the upper right of the main product page, and then click the Blueprints tab on the upper left.

   The Blueprints pane opens, listing all existing blueprints.

2. Select Create Blueprint from the Table Actions drop-down list in the Blueprints pane.

   The Create Blueprint wizard opens.

3. Complete the fields (see page 58) on the Blueprint page Component Blueprint pane, and then click Next.

   The Discovery Methods page opens. It displays the File Indicators pane and the Add New Search Options pane.

4. Complete the Search Options fields (see page 60)on the Add New Search Options pane.

5. Click Add Directory/File, and then complete the File Indicators fields (see page 60) in the Add New File pane.

6. Click the Registry Indicators link.

   The Registry Indicators pane displays with Registry Indicators expanded and the \HKEY_LOCAL_MACHINE element selected. The \HKEY_LOCAL_MACHINE pane displays the Search Options fields.

7. Complete the Search Options fields (see page 62) in the \HKEY_LOCAL_MACHINE pane, and then click Save.

8. Click Add Registry Value/Key.

9. Complete the Registry Indicators fields (see page 62) in the Add New Registry Value/Key pane.

10. Click the Network Probes link.

    The Network Probes pane displays with the Network Probes element selected. The Add New Network Probe pane displays the Network Probe fields.

    On 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

11. Complete the Network Probe fields (see page 63), and then click Next.

    The Discovery Verification Rules page displays with the Discovery Verification Rules element selected in the left pane. The Add New Discovery Verification Rule 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.

12. Complete the Discovery Verification Rule fields (see page 64), and then click Next.

    The product saves the Discovery Verification Rule values, and then displays the Management page. The $(Root) folder is selected in the File Management pane, and the $(Root) pane displays the File Management Options fields. The 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 Management page lets you refine the managed component view.

13. Complete the File Management Options fields (see page 69) in the Management page $(Root) pane.

14. (Optional) Click Add Directory, complete the Directory fields (see page 70) on the Add New Directory pane, and then click Save.

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

15. (Optional) Click Add File, complete the File fields (see page 70) on the Add New File pane, and then click Save.

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

16. (Optional) Select a node in the File Management pane and repeat Steps 14 and 15 as appropriate to create subdirectories and other nested files.

    The product adds the new elements below the selected node in the File Management pane.

17. Click the File Filters and Attributes link.

    The File Filters and Attributes pane displays the directory and file structure that you created in Steps 14 through 16 below the $(Root) folder.

18. Define filters and attributes for the blueprint file.

19. Click the Registry Management link, complete the Registry Management fields (see page 72), and then click Save.

20. (Optional) Click Add Key, complete the Key fields on the Add New Key pane, and then click Save.

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

21. (Optional) Click Add Value, complete the Value fields on the Add New Value pane, and then click Save.

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

22. (Optional) Select a node in the Registry Management pane and repeat Steps 20 and 21 as appropriate to create other nested keys and values.

    The product adds the new keys or values below the selected node in the Registry Management pane.

23. Define registry filters and attributes for the blueprint. (see page 25)

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

24. Click the Data Management link.

    The left pane displays the Data Management folder, and the right pane displays the Database page.

25. Complete the fields (see page 76) on the Database page, and then click Save.

    The database appears in the Data Management pane.

26. Click Next.

    The Component Parameters and Variables page opens.

27. Complete the fields on the Component Parameters and Variables page, and then click Save.

28. Click Next.

    The Configuration - File Parsing page opens.

29. Complete the fields (see page 88) on the Configuration - File Parsing page, and then click the Configuration Executables link.

30. Complete the fields (see page 89) on the Configuration Executables page, and then click Save.

31. Click the Configuration Data link, and then complete the Database field, which defines the database that the blueprint uses. The drop-down list displays the databases that you created in Step 23.

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

33. Complete the fields (see page 94) on the Add Query pane, and then click Save. The Configuration Data tree in the left pane shows the query.

34. Click the File Structure Data link, complete the fields (see page 95) on the File Structure Class tab, and then click Save. The File Structure Class tree in the left pane shows the structure class.

35. Click the Precedence tab, click Add Group or Add Parameter, and then complete the displayed fields (see page 96).

36. Click Next, complete the fields (see page 99) on the Macros page, and then click Next.

    The Component Grouping Options page opens. The page contains options that let you nest components in a service for display to emphasize the relationships between them. For example, when a component depends on subordinate components in the primary component file system root, you can use nesting to enforce the parent-child relationship.

    Oracle databases, for example, typically install a Java Runtime Engine and an Apache Web server in the installation directory. The product expresses the relationship between the utility components and the Oracle database by nesting the JRE and Apache in the Oracle component.

37. Complete the fields (see page 100) on the Component Grouping Options page, and then click Finish.

    The product creates the blueprint, which then appears in the Blueprint table.

Define Blueprint File Filters and Attributes

Follow these steps:

1. Select the $(Root) folder.

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

2. (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 preceding *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.

3. (Optional) Repeat Step 2 for the files that the Files table lists.

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

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

   The product adds your selections to the Selected Categories column.

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

   The product adds your selections to the Selected Filters column.

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

8. (Optional) Click the Manage Filters tab.

   The Manage Filters tab lists the sub folders and files of the selected folder. The Manage Filters tab lets you apply or remove the Never Run Change Detection, and Time Variant filters to specific folders, its sub folders, and files.

9. (Optional) Select the sub folders and files from the list, and then select one of the following actions from the Select Actions drop-down:

   - Set Never Run change Detection

     Apply the Never Run change Detection filter to the selected folders, its sub folders, and files that are defined in the File Filters and Attributes pane.

   - Set Time Variant

     Apply Time Variant filter to selected folders, its sub folders, and files that are defined in the File Filters and Attributes pane.

   - Remove Never Run change Detection

     Remove the Never Run change Detection filter from selected folders, its sub folders, and files that are defined in the File Filters and Attributes pane.

   - Remove Time Variant

     Remove the Time Variant filter from selected folders, its sub folders, and files that are defined in the File Filters and Attributes pane.

   A message confirms whether the Never Run Change Detection and Time Variant filters are updated for the folder, its sub folders, and files.

   Note: If you add new sub folders or files to a folder, the existing filter does not apply to the newly added sub folder or files. Apply a new filter to the newly added sub folder or file when required.

10. 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 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.

11. Complete the fields (see page 70) on the Rules tab.

Define Blueprint Registry Filters and Attributes

Follow these steps:

1. Click the Registry Filters and Attributes link.

   The Registry Filters and Attributes pane displays the $(RegistryRoot) folder and the $(RegistryRoot) pane displays the Precedence tab. The tab shows existing keys and values in the corresponding tables.

2. (Optional) Select a row 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 18.

3. Click Add Key, and then complete the Key pane Name (regex) and Description fields.

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

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

6. Complete the remaining fields in the Key pane, and then click Save.

7. In the Registry Filters and Attributes pane, select the key to which to assign a value.

8. In the right pane, click Add Value, and then complete the Value pane Name (regex) and Description fields.

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

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

11. Complete the remaining fields in the Value pane, and then click Save.

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

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));)

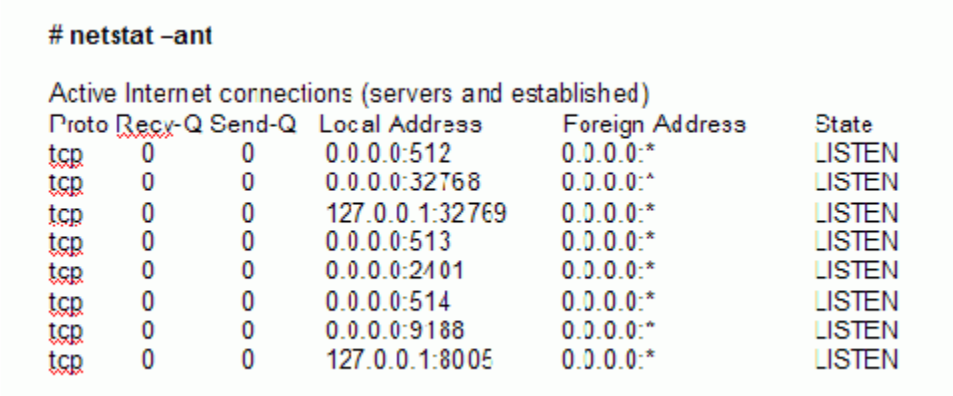
Defines and calls functions.

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

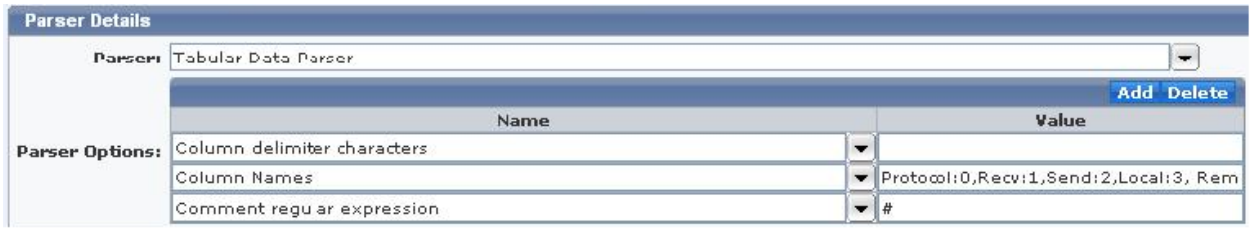
CA Configuration Automation provides a Tabular Data Parser that interprets and parses any form of tabular data within configuration files or executables. In addition, you can specify parser options that control the layout of rows and columns within the tabular data set, assign names to columns, eliminate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

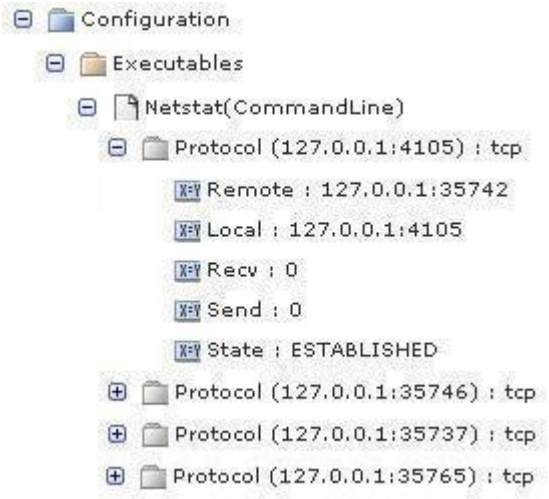
Defines the assignment of names to individual columns of data. The format of the field is a comma-delimited list of names and column index numbers. Column indexing starts at zero. For example:

"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:

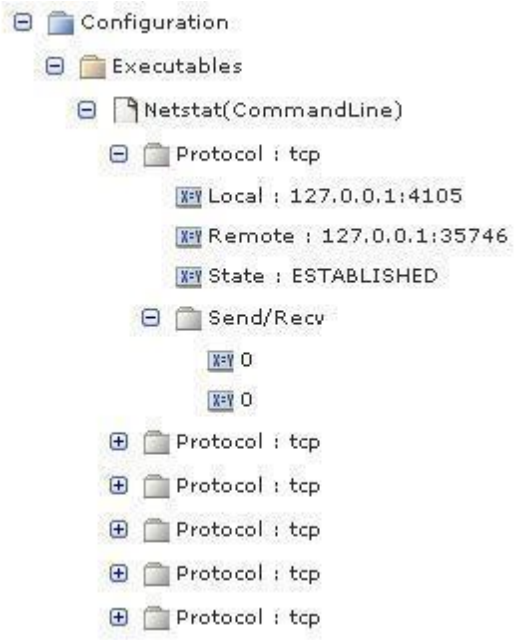


Note: The Structure Class used to interpret the parsed data in this example specifies Local as the qualifier for Protocol.

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

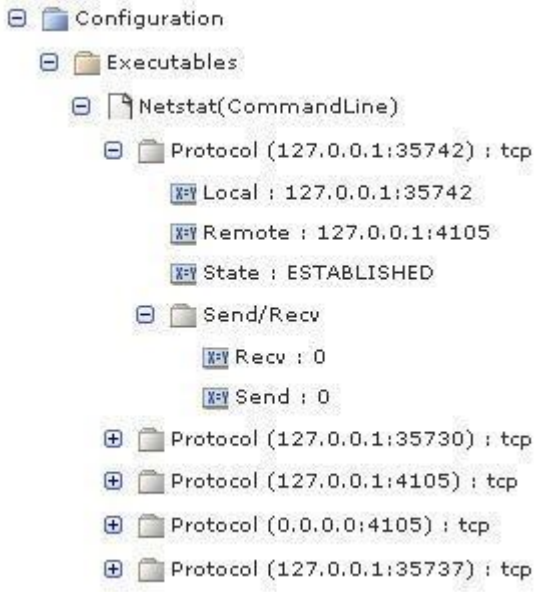
The data would be displayed as:



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

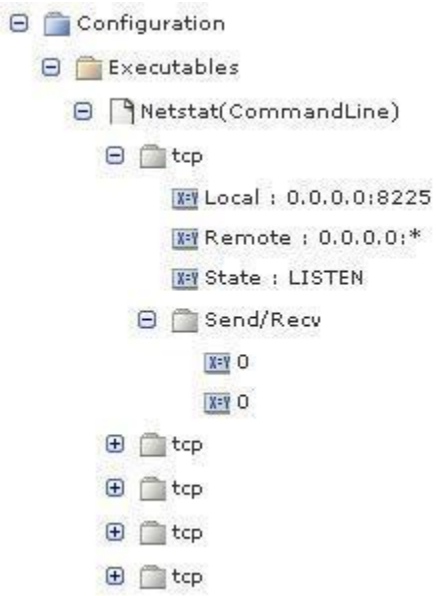
Columns 3 through 5, 7, and 9 through 11

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	7381	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.

Appendix A: Blueprint Wizard UI Reference

This section contains the following topics:

Blueprint Page: Component Blueprint Fields

The Component Blueprint pane on the Blueprint wizard Blueprint page contains the following fields:

Component Blueprint Name

Defines a unique name for the blueprint.

Limitations: The blueprint name cannot contain the following characters:
< > ; : " ' * + = \ / | ?

Component Version

Defines the software component version (release number).

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

Default: *.*

Blueprint Version

Defines the blueprint version, in your preferred format.

Default: 1.0.0

Discovery

Specifies whether discovery is enabled or disabled for the blueprint.

Default: Enabled

Description

Describes the blueprint item. The blueprint Details page displays the description.

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 Cisco routers and network switches that use IOS.

  You can always add or remove operating system-specific files. 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.

Default: Any

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.

Default: Custom Components

Discovery Methods Page: Search Options Fields

The Add New Search Options pane on the Blueprint wizard Discovery Methods page contains the following Search Options fields:

Search From

Defines where the search starts. 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 number of folders or directories the search includes.

Default: 10

Component Always Found

Specifies whether the process always considers the component as discovered.

Yes

The product does not search for the indicators. If you define a $(Root) parameter as a constant, the product manages files under the root. Select Yes to define and manage components in services that the product cannot or does not need to discover.

No

The product searches for the indicators.

Default: No

Discovery Methods Page: File Indicators Fields

The Add New File pane on the Blueprint wizard Discovery Methods page contains the following fields:

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 the product adds a File or a Directory to the blueprint.

Default: File

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

Path From Root

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 locates 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 matching the full expression. Use wildcard characters in the partial path for intermediate directories with unknown variable names. For example, *\conf
- You can use the * or ? wildcard characters in the directory names. For example: *conf, \conf*, /*bin, /bin*, /b*n, and /bi?

Depth From Root

Defines the number of folders or directories the discovery includes, beginning from the search root.

Discovery Methods Page: Registry Indicators Search Options Fields

The \HKEY_LOCAL_MACHINE pane on the Blueprint wizard Discovery Methods page for Registry Indicators contains the following Search Options fields:

Search From

Defines where the search starts. 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 number of folders or directories the search includes.

Default: 10

Component Always Found

Specifies whether the process always considers the component as discovered.

Yes

The product does not search for the indicators. If you define a $(Root) parameter as a constant, the product manages files under the root. Select Yes to define and manage components in services that the product cannot or does not need to discover.

No

The product searches for the indicators.

Default: No

Discovery Methods Page: Registry Indicators Fields

The Add New Registry Value/Key pane on the Blueprint wizard Discovery Methods page for Registry Indicators contains the following Registry Indicators fields:

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 the product adds a File or a Directory to the blueprint.

Default: File

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

Path From Root

Defines the path from the search root.

Depth From Root

Defines the number of folders or directories the discovery includes, beginning from the search root.

Discovery Methods Page: Network Probe Fields

The Add New Network Probe pane on the Blueprint wizard Discovery Methods page for Network Probes contains the following Network Probe fields:

Name

Defines a name for the network probe.

Description

Describes the blueprint item.

Primary Ports

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

Alternate Ports

Defines a comma-delimited list of port numbers from which to collect responses if the primary ports fail.

Probe

Describes an optional probe that is sent from CCA to the remote service after the port communications have been established. The probe is to verify that the service or component exists.

Regex Match

Defines the attribute that the product 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).

Default: TCP

Discovery Verification Rules Page: Discovery Verification Rule Fields

The Add New Discovery Verification Rule pane on the Blueprint wizard Discovery Verification Rules page contains the following Discovery Verification Rule fields:

Name

Defines a name for the discovery verification rule.

Description

Describes the blueprint item.

Directive Type

Defines which of the following directive types the product uses to:

- Extract the values from elements that are managed in a service
- Retrieve the 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:

- Define the fixed Root and the RegistryRoot parameters when the component installation location is not ambiguous.
- Combine data from multiple sources by assembling variables and fixed text in a complex string.
- Expose component constants (for example, 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:

- Run a query that extracts a column value from a database row.
- Run a stored procedure that changes the managed server or extracts a value.
- Run a query that extracts a result set and displays it in tabular form in a macro step directive.

Get File

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

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

Get SNMP

Retrieves the value at a management information base (MIB) address from an SNMP agent, and (optionally) sets the directive value from the results.

Match File Name

Lists all files and directories at a designated location in a managed server file system, and (optionally) extracts the directive value from the list.

Match Registry Name

(Windows Only) Lists registry key and registry value names at a location in the registry tree, and (optionally) extracts the directive value from the list.

Match Registry Name Get Data

(Windows Only) Lists all registry key and registry value names at a location in the registry tree.

(Optional) 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. You can (optionally) send a probe to the port, and collect a response from the socket as the directive value.

Registry

(Windows only) Retrieves the data value that is associated with a registry key or value, and (optionally) defines the directive value from the filtered results.

Remote Execution

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

- Access the configuration information that is only available from output (for example, operating system configurations).
- Access transient data such as memory, network, or CPU statistics.
- Access custom scripts and tools and import the output to the blueprint.
- Run utilities and scripts to update a managed server.

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 to 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.

Default: Constant

Stage

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

Late

The verification directive requires a Component parameter value for the variable substitution. The Component parameter value is the parameter name that is defined in the Component Parameters and Variables tab. 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

Defines a 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, the 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 the product 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*)

Defines 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 you use the variable substitution 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:

- Return paren(0) if the value matches the expression.
- Return no value if the value and the expression do not match.

Paren

Defines which subexpression the product returns on match if a regular expression includes a parenthesized subexpression.

Default: 0 (if you specify no Paren value).

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 text case 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 a unique name that the translation applies to the directive output. The product precedes the specified Translate value with $CCTranslation$__ at run time so the value is identifiable in the database as a translation. If no translation matches, the product retains the original directive result.

The directive values that the product extracts from the configuration files and the registry is sometimes cryptic strings or integers that belong to an enumeration. Each value corresponds to a configuration state, but the value does not interpret the state. The product can translate these values to meaningful strings so they are clear when it displays them in the blueprint.

Map values in the product Database to refer to the translation name and trigger the translation of *from values* to *to values*. The following example maps the translation $CCTranslation$__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>
```

The product does not associate translations with a specific blueprint. To avoid conflict with other translations, ensure that each translation has a unique name.

Note: The product user interface does not currently support defining translation tables. Create the files in CA Configuration Automation data load format (as the example shows), and then load them to the database with the data loader utility.

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 the product inserts in another string to produce the appropriate result for display or for a subsequent operation.

The insertion location can be any string that contains $(VALUE) (all uppercase letters). 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 is 1.3.export.

Modifier

Modifies the discovery results. Select either the Does Host Use Agent or Is Alterpoint Device modifiers.

Modifier Parameters

Defines the parameters that the product applies to the specified modifier.

Management Page: File Management Options Fields

The $(Root) pane on the Blueprint wizard Management page contains the following File Management Options fields:

Managed Depth

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

Retrieve configured maximum files (50,000)

Specifies how many files the discovery operation retrieves.

Selected: The discovery operation retrieves up to 50,000 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.

Management Page: Directory Fields

The Add New Directory pane on the Blueprint wizard Management page contains the following Directory fields:

Name (posix)

Defines a name for the element to add. You can use wildcard characters (*) to define the name with the POSIX pattern-matching syntax (for example, Agent_*).

Path From Root

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

Management Page: Directory Fields

The Add New File pane on the Blueprint wizard Management page contains the following File fields:

Name (posix)

Defines a name for the element to add. You can use wildcard characters (*) to define the name with the POSIX pattern-matching syntax (for example, Agent_*).

Path From Root

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

Filters and Attributes Page Rules Tab Fields

The Rules tab on the Component Blueprint wizard File Filters and Attributes page contains the following fields:

Name

Defines a name for the rule.

Description

Describes the blueprint item.

Documentation URL

Specifies a URL where the rule 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 that is taken as the reference for the Operation. The constraint type, operation, and value define a match criteria for the 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 that the rule failed.

- Information
- Warning
- Error
- Critical

Registry Management Fields

The $(RegistryRoot) pane on the Blueprint wizard Management page contains the following Registry Management Options fields:

Managed Depth

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

Retrieve configured maximum elements (50,000)

Specifies how many elements the discovery operation retrieves.

Selected: The discovery operation retrieves up to 50,000 elements.

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

Maximum Elements

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

Registry Filters and Attributes Page Add Key Fields

The Component Blueprint wizard Registry Filters and Attributes page contains the following Add Key fields:

Default

Defines the default registry key value.

Interpret As

Specifies which of the following entities the product interprets the key as:

- 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.

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 discovered registry key is shown or hidden. Also, specifies whether the default value defined in the Blueprint UI is shown or hidden. The value is masked and encrypted when you select the Visibility value as Hide Value.

Show Value

Displays the default value, and discovered registry key.

Hide Value

Hides the default value in the Blueprint UI, and discovered registry key. Displays ******* instead of the values. Hide the value if:

- The value is confidential (for example, a password).
- The value is binary.
- The value is too long to display.

The product encrypts the hidden registry key value in the database. They are not displayed or viewable in the UI.

Note: To ensure the security of this field, reenter the value if you later decide to show it.

Interpreted Server

Defines the target server information that is available in any other component parameter. To define the parameter, use variable substitution.

Interpreted Target Instance

Defines the target database instance or application instance that is available in any other component parameter. To define the parameter, use variable substitution.

Interpreted Application

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

Registry Filters and Attributes Page Value Details Fields

The Component Blueprint wizard Registry Filters and Attributes page contains the following Value Details fields:

Default

Defines the default registry key value.

Interpret As

Specifies which of the following entities the product interprets the key as:

- 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.

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 discovered registry value is shown or hidden. Also, specifies whether the default value defined in the Blueprint UI is shown or hidden. The value is masked and encrypted when you select the Visibility value as Hide Value.

Show Value

Displays the default value, and discovered registry value.

Hide Value

Hides the default value in the Blueprint UI, and discovered registry value. Displays ******* instead of the values. Hide the value if:

- The value is confidential (for example, a password).
- The value is binary.
- The value is too long to display.

The product encrypts the hidden registry key value in the database. They are not displayed or viewable in the UI.

Note: To ensure the security of this field, reenter the value if you later decide to show it.

Interpreted Server

Defines the target server information that is available in any other component parameter. To define the parameter, use variable substitution.

Interpreted Target Instance

Defines the target database instance or application instance that is available in any other component parameter. To define the parameter, use variable substitution.

Interpreted Application

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

Database Page Fields

The Component Blueprint wizard Database page contains the following fields:

Name

Defines a name for the database.

Description

Describes the blueprint item.

Database

Defines the database access description.

User

Defines the account name of a user that can access the database.

Password

Defines the password that is associated with the specified user account.

Server

Defines the server where the database is installed.

Port

Defines the listening port of the database host server.

DB Type

Specifies the type of database in use, from the following options:

- 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
- SQL_SERVER2K8_CONTEXT

Environment

Defines the environment variables that the agent sets before it runs the command.

The product runs the remote command in the CA Configuration Automation Agent environment. If the agent does not have the environment setup that is required to run a command, define the 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 supports the variable substitution method.

Component Parameters and Variables Page Fields

The Component Blueprint wizard Component Parameters and Variables page contains the following fields:

Name

Defines a parameter substitution expression in the following form:

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

VariableName

Defines the name of a discovery parameter that is defined in the component. By default, the Name value 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 on their own.
- You can define recursive multiple substitutions in the same expression. For example, a substitution value can be a string in the parameter expression. If it is, the blueprint evaluates it recursively.

Example:

If the Discovery parameters have the following values:

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

the following parameter substitution expression:

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

returns the following results after evaluation:

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

Description

Describes the blueprint item.

Folder

Defines the item location.

Selected Categories

Defines which of the following categories the product uses to organize the elements. When you double-click an item in the Available Categories column, it moves to the Selected Categories column.

Administration

Defines administrative settings relative to general component availability and management. Examples of administrative settings include:

- How to back up elements.
- When to delete a cache.
- How many times to retry an action.

Configuration

Defines component configuration settings that are not related to the Administration, Log and Debug, Network, or Performance settings. Examples of configuration settings include a component alias or default web page.

Documentation

Defines the elements that document the component behavior or provide information to users. For example, guides, readme files, FAQs, or online help pages.

Log And Debug

Defines the elements that let the user set log locations, log levels, debug output, or diagnostic variable types.

Network

Defines the elements that represent a network-related component setting (for example, the SNMP port settings). If an element can be categorized as both Network and Security (for example, enabling LDAP Authentication), set the category to Security.

Other

CA internal use only.

Performance

Defines the performance settings, which are generally a specific subset of configuration parameters. For example, number of threads or number of concurrent users.

Product Info

Defines general (static) product information (for example, licensing, installation location, vendor, or module name).

Resources

Defines component resources (for example, storage, memory and cache allocation or size, or CPU).

Note: The static Resource category is different from the Transient category, which defines real-time information.

Security

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

Transient

Defines the elements that regularly change. For example, server states (up, down, running, or stopped), current number of connected clients, current number of threads, or current disk utilization.

Versioning and Patches

Defines the elements that indicate the product version or patch levels.

Selected Filters

Defines which of the following elements the product filters from the Compare operations. When you double-click an item in the Available Filters column, it moves to the Selected Filters column.

Component Specific

Filters the elements that are specific to a single component instance (for example, installation root and Service Server Name). Excludes component-specific elements that are already known to be different from Change Detection operations and results.

Service Specific

Filters the elements that are specific to a service (for example, server names and installation roots). Excludes service-specific elements that are already known to be different from Change Detection operations and results.

Server Specific

Filters the elements that are specific to a single server (for example, the Server name and IP address). Excludes server-specific elements that are already known to be different from Change Detection operations and results.

Never Run Change Detection

Identifies the elements to exclude permanently from Change Detection operations and results. Examples include temporary directories, log files in the managed folder, or elements that are known to be transient.

Never Run Rule Compliance

Identifies the elements to exclude permanently from Rule Compliance operations and results because they are inconsequential or variable. Examples include temporary directories, known old configuration files, templates, or example files.

Time Variant

Filters the elements that are known to change over time, but not necessarily across servers or across services. Examples include log files, process start times, and registry event counters. Excludes time-variant parameters that are already known to be different from Change Detection operations and results.

File Size Variant

Filters the files based on the file size in the Change Detection Operations only when the Time Variant filter is not selected.

Weight

Specifies the relative importance of an element:

- Low
- Medium
- High

Default: Medium (elements for which you assign no weight)

Directive Type

Defines which of the following directive types the product uses to:

- Extract the values from elements that are managed in a service
- Retrieve the 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:

- Define the fixed Root and the RegistryRoot parameters when the component installation location is not ambiguous.
- Combine data from multiple sources by assembling variables and fixed text in a complex string.
- Expose component constants (for example, the vendor name) as parameters.

Configuration

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

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:

- Run a query that extracts a column value from a database row.
- Run a stored procedure that changes the managed server or extracts a value.
- Run a query that extracts a result set and displays it in tabular form in a macro step directive.

Get File

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

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

Get LDAP

Retrieves named data sets from a Directory Server, and (optionally) extracts the directive value from the output.

Get SNMP

Retrieves the value at a management information base (MIB) address from an SNMP agent, and (optionally) sets the directive value from the results.

Match File Name

Lists all files and directories at a designated location in a managed server file system, and (optionally) extracts the directive value from the list.

Match Registry Name

(Windows Only) Lists registry key and registry value names at a location in the registry tree, and (optionally) extracts the directive value from the list.

Match Registry Name Get Data

(Windows Only) Lists all registry key and registry value names at a location in the registry tree. You can (optionally) set the directive value from the data value of the selected names.

Network Probe

Defines the parameters of a TCP socket opened to a remote service server and port. You can (optionally) send a probe to the port, and collect a response from the socket as the directive value.

Registry

(Windows only) Retrieves the data value that is associated with a registry key or value, and (optionally) defines the directive value from the filtered results.

Remote Execution

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

- Access the configuration information that is only available from output (for example, operating system configurations).
- Access transient data such as memory, network, or CPU statistics.
- Access custom scripts and tools and import the output to the blueprint.

Run utilities and scripts to update a managed server.

Default

Defines a fixed string or a string that contains one or more variable substitutions. If the directive runs and the product cannot determine a value, the product uses the default value instead of an undefined or zero-length value.

Persistence

Specifies whether a parameter persists in a service.

Always Persist

The product saves the parameter and displays it in the service tree view.

Never Persist

The product does not save the parameter only uses it for discovery.

Persist If Not Null

The product saves the parameter and (if the parameter has a value) displays it in the service tree view.

Default: Always Persist

Visibility

Specifies whether the component shows or hides the component parameter and its value. Also, specifies whether the default value defined in the Blueprint UI is shown or hidden. If you hide the value, the value is encrypted.

Show Value

Displays the default value, and component parameter and its value.

Hide Value

Hides the default Value in Blueprint UI and the component parameters value in the component viewer UI and displays ******** instead of the respective values. Hide the component parameter value and default value if:

- The value is confidential (for example, a password).
- The value is binary.
- The value is too long to display.

The product encrypts the hidden component parameters value in the database. They are not displayed or viewable in the UI.

Note: To ensure the security of this field, reenter the value if you later decide to show it.

Hide Element

Hides the parameter. Select Hide Element so the value is available for variable substitution, but is not displayed in the service tree view.

Default: Show Value

Value Case

Specifies whether the product considers text case 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.

Interpret As

Defines a hint about the string format of a configuration parameter and how the associated component uses it. The product uses context-sensitive parsers to inspect interpreted parameter values. The parsers let the product extract multiple subvalues from complex parameter strings.

Default: blank (no interpretation)

Relationship Key

Specifies whether the product determines and assigns relationships according to the Interpret As value.

To establish relationships, set the Interpret As value and then set the Relationship Key field to **Yes**. Not all interpretations can define relationships.

Default: No

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

Defines the target server information that is available in any other component parameter. To define the parameter, use variable substitution.

Interpreted Target Instance

Defines the target database instance or application instance that is available in any other component parameter. To define the parameter, use variable substitution.

Interpreted Application

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

Translate

Defines a unique name that the translation applies to the directive output. The product precedes the specified Translate value with $CCTranslation$__ at run time so the value is identifiable in the database as a translation. If no translation matches, the product retains the original directive result.

The directive values that the product extracts from the configuration files and the registry is sometimes cryptic strings or integers that belong to an enumeration. Each value corresponds to a configuration state, but the value does not interpret the state. The product can translate these values to meaningful strings so they are clear when it displays them in the blueprint.

Map values in the product Database to refer to the translation name and trigger the translation of *from values* to *to values*. The following example maps the translation $CCTranslation$__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>
```

The product does not associate translations with a specific blueprint. To avoid conflict with other translations, ensure that each translation has a unique name.

Note: The product user interface does not currently support defining translation tables. Create the files in CA Configuration Automation data load format (as the example shows), and then load them to the database with the data loader utility.

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 the product inserts in another string to produce the appropriate result for display or for a subsequent operation.

The insertion location can be any string that contains $(VALUE) (all uppercase letters). 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 is 1.3.export.

Modifier

Modifies the discovery results. Select either the Does Host Use Agent or Is Alterpoint Device modifiers.

Modifier Parameters

Defines the parameters that the product applies to the specified modifier.

Configuration - File Parsing Page

The Configuration - File Parsing page contains the following fields:

Name (regex)

Defines the name of the configuration file that the product locates and parses during discovery. You can use wildcard characters (for example, ^ or $) to define the name as per regex naming conventions.

Display Name

Defines the name of the configuration file that the product displays after discovery.

Description

Describes the blueprint item.

File Type

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

Configuration Executables Page

The Configuration Executables page contains the following fields:

Name

Defines a name for the configuration executable. This name appears under Executables in the Configuration folder. The name must be unique in the Executables folder.

Description

(Optional) Describes the blueprint item. The product displays the description as a tool-tip over the directive name in the blueprint and over the parameter name in the tree views.

Directive Type

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

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

Default: Constant

File

Defines the name of a configuration file that is 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, the product selects all matching files in the Configuration folder. If multiple files match, the product selects the file closest to the root of the managed file system. The product allows variable substitution.

Path

Defines the path to the configuration file. The product does not support wildcard characters or pattern matching, but it allows variable substitution. The path can either be absolute (for example, $(Root)/conf) or relative (for example, /conf) to the root of the managed file system. If the configuration file is at the configuration file system root, you can define Path as $(Root) or you can leave it undefined.

Parameter

Defines the name of the configuration parameter in the file. The product does not support wildcard characters or pattern matching, but it allows variable substitution. You can leave the Parameter attribute undefined if the value belongs to a Group file block. In that case, specify only the Group Path attribute.

Regex

Specifies whether to filter the named value with a regular expression if the product finds it.

Paren

Defines which subexpression the product returns on match if a regular expression includes a parenthesized subexpression.

Default: 0 (if you specify no Paren value).

Group Path

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

Because a single file often contains the same name many times, use a unique Group Path to differentiate a specific value from the matching names. For example, an XML configuration file can have the following format, where the tag server and the buildDate and type attributes 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>
```

To identify children that must match for the product to follow the path, use a comma-separated list of Group Path names (or names and values). For example, to extract the buildDate value for server wxp124, define the following values:

Parameter

buildDate

Group Path

configuration/server,name=wxp124/setup

Extracting the buildDate value from the server wxp123 auxiliary data only requires the name because it is the only server tag with an auxiliary attribute:

Parameter

buildDate

Group Path

configuration/server,auxiliary/setup

You can qualify any or all of the elements in a Group Path and you can specify multiple qualifiers for each element. If you specify multiple qualifiers, each qualifier must match to select a path. The product allows variable substitution.

If a structure class for the configuration file defines group blocks qualifiers, the product allows an alternate syntax. For group blocks, you can surround the qualifier value with parentheses and append it to the block name. For example, to extract the value of buildDate from server wxp124 where the attribute name is a Qualifier Child, specify the following values:

Parameter

buildDate

Group Path

configuration/server(wxp124)/setup

The value in parentheses matches the qualifier of the named group block. Either a Qualifier Child value or a constant Qualifier in the structure class defines the qualifier, or the qualifier is the group value.

If you are not sure about the parameter location in the file, you can specify multiple paths. Separate two or more specified paths with a pipe (|). The product tries the paths one at a time, from left to right, until a match it finds a match. For example:

Parameter

buildDate

GroupPath

configuration/server(abc)/setup|configuration/server(xyz)/setup

Empty Is Null

Specifies whether to override the default product behavior when the named configuration parameter in the file has no value. The default behavior is to set the value to an empty, zero-length value. The Empty Is Null field lets you distinguish between a directive that returns an empty value and a directive that returns no value.

Yes

Uses the value in the Default field.

No

Does not use the value in the Default field.

Default: No

Default

Defines a fixed string or a string that contains one or more variable substitutions. If the product cannot determine a value when it runs the directive, the product uses this value instead of an undefined or zero-length value.

In some cases (for example, if a Component Blueprint plug-in can set a directive), you do not want the Default value to replace the directive. To keep the Default value from replacing the plug-in value, set it to **Cohesion.PLACEHOLDER**.

Value Case

Specifies whether the product considers text case 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 a unique name that the translation applies to the directive output. The product precedes the specified Translate value with $CCTranslation$__ at run time so the value is identifiable in the database as a translation. If no translation matches, the product retains the original directive result.

The directive values that the product extracts from the configuration files and the registry is sometimes cryptic strings or integers that belong to an enumeration. Each value corresponds to a configuration state, but the value does not interpret the state. The product can translate these values to meaningful strings so they are clear when it displays them in the blueprint.

Map values in the product Database to refer to the translation name and trigger the translation of *from values* to *to values*. The following example maps the translation $CCTranslation$__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>
```

The product does not associate translations with a specific blueprint. To avoid conflict with other translations, ensure that each translation has a unique name.

Note: The product user interface does not currently support defining translation tables. Create the files in CA Configuration Automation data load format (as the example shows), and then load them to the database with the data loader utility.

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 the product inserts in another string to produce the appropriate result for display or for a subsequent operation.

The insertion location can be any string that contains $(VALUE) (all uppercase letters). 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 is 1.3.export.

Modifier

Modifies the discovery results. Select either the Does Host Use Agent or Is Alterpoint Device modifiers.

Modifier Parameters

Defines the parameters that the product applies to the specified modifier.

Parser Details

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

Add Query Pane

The Component Blueprint wizard Add Query pane contains the following fields:

Name

Defines which directive queries a database on a managed server, and (optionally) extracts a value from the output data. This directive is commonly used to:

- Run a query that extracts a column value from a database row.
- Run a stored procedure that modifies the managed server or to extract a value.
- Run a query that extracts a result set and displays it in tabular form in a directive macro step.

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

Description

(Optional) Describes the blueprint item. The product displays the description as a tool-tip over the directive name in the blueprint and over the parameter name in the tree views.

Query Type

Specifies one of the following query types that direct the product how to run the query and whether the query returns data:

Select

The query is an SQL select statement that returns data.

Insert

The query is an SQL insert statement that returns no data.

Delete

The query is an SQL delete statement that returns no data.

Update

The query is an SQL update statement that returns no data.

Others

The query is an SQL alter table or other DDL statement that returns no data.

Stored Procedure

The query string invokes a stored procedure that can return data.

Query

Defines an SQL query or stored procedure name. Query syntax can vary depending on the database type, and variable substitution is allowed. The product returns all rows that the query or stored procedure finds. To filter the result set, use the column and regular expression attributes.

Max Rows

Defines the maximum number of rows that a query returns.

Default: 5000

Primary Columns

Defines the name of the column or an aliased column in the returned result set. The product takes the value in the named column as the directive value. If the query returns more than one row, the product uses the named column in the first row.

Structure Class

Defines the hierarchical collection of groups and parameters that the product uses to map metadata to the name/value pairs that the parser retrieves.

Note (Edit Mode only)

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

- If you select Show All Notes, a new page lists all notes. To change the table sort order, click any table heading.
- If you select New Note, the following fields appear:

  Name

  Defines a name for the note.

  Text

  Defines the note text.

If you select an existing note, the Notes field shows the note name with the note text below the name. You can update, delete, or cancel the selected note.

File Structure Class Tab

The File Structure Class tab contains the following fields:

Name

Defines a name for the structure class. The Blueprint list and (unless you specify a Display Name) the tree view show the specified structure class name.

Version

Defines the structure class version of the structure class. The Blueprint list and (unless you specify a Display Name) the tree view show the specified version.

Display Name

Defines the class name that the Component Tree view shows if it differs from the Name and Version fields. For example, you can specify a Display Name value that is more descriptive of the class function.

Description

(Optional) Describes the blueprint item. The product displays this description as a tool-tip over the class name in the Blueprint tree view.

Allow Remediation Jobs

Specifies whether the product can modify the changeable blueprint elements as specified in a remediation job.

Default: Yes

Parser

Defines the parser to use for files of the defined type. The product includes a library of predefined lexical analyzers (lexers) and parsers that let the product analyze most common configuration file formats.

File Structure Class Group and Parameter Fields

The Add Group and Add Parameter panes on the Precedence tab contain the following fields:

Name

Defines name for the precedence group.

Description

Describes the blueprint item.

Available Categories

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

Available Filters

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

Weight

Specifies the relative importance of an element:

- Low
- Medium
- High

Default: Medium (elements for which you assign no weight)

Data Type

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

Valid Values

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

Default Value

Defines a fixed string or a string that contains one or more variable substitutions. If it cannot determine a value at run time, the product uses the Default Value instead of an undefined or zero-length value.

If a Component Blueprint plug-in can set the directive value, you can set Default Value to Cohesion.PLACEHOLDER to keep the specified Default Value from replacing the plug-in value.

Visibility

Specifies whether the component shows or hides the configuration parameter or group value. Also, specifies whether the default value defined in the Blueprint UI is shown or hidden. If you hide the value, the value is encrypted.

Show Value

Displays the default value, and configuration parameter or group value.

Hide Value

Hides the default value in Blueprint UI and configuration parameter or group value in the component viewer UI.

- The value is confidential (for example, a password).
- The value is binary.
- The value is too long to display.

The product encrypts the hidden configuration parameters or group value in the database. They are not displayed or viewable in the UI.

Note: To ensure the security of this field, reenter the value if you later decide to show it.

Default: Show Value

Value Case

Specifies whether the product considers text case 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.

Interpret As

Defines a hint about the string format of a configuration parameter and how the associated component uses it. The product uses context-sensitive parsers to inspect interpreted parameter values. The parsers let the product extract multiple subvalues from complex parameter strings.

Default: blank (no interpretation)

Relationship Key

Specifies whether the product determines and assigns relationships according to the Interpret As value.

To establish relationships, set the Interpret As value and then set the Relationship Key field to **Yes**. Not all interpretations can define relationships.

Default: No

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

Defines the target server information that is available in any other component parameter. To define the parameter, use variable substitution.

Interpreted Source Instance

Defines the source database instance or application instance.

Interpreted Target Instance

Defines the target database instance or application instance that is available in any other component parameter. To define the parameter, use variable substitution.

Interpreted Application

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

Expected Value

Defines the expected value of the parameter or group.

Value

Defines a unique value for the group in the file.

Qualifier Child (Add Group only)

Defines the parameter that the product uses as a qualifier under the group.

Macros Page

The Macros page contains in the following fields:

Name

Defines a name for the macro.

Description

Describes the blueprint item.

Folder

Defines the item location.

Diagnostic

Specifies whether the product uses the macro to diagnose issues.

Default: No

Read Only

Specifies whether the macro can modify the target system.

Note: To run a read-only macro, the user must have Server View or Service View permissions.

Default: No (the macro cannot modify the target system).

Finish Page

The wizard page contains the following fields:

Allow this blueprint to contain other components

Specifies whether the blueprint can contain other components.

Allow this blueprint to nest in other components

Specifies whether other components can contain the blueprint.

Note: By default, the Allow this blueprint to contain other components, and Allow this blueprint to nest in other components checkboxes are enabled.

Allow this blueprint to contain only named components

Specifies whether the blueprint can contain only named components.

This blueprint will not use nesting

Specifies whether the blueprint can use nesting.

Refresh Order

Specifies the position that this component refreshes relative to all others on a specific server. The product refreshes components on each server sequentially and refreshes components on other servers in parallel.

Refresh order is important when components depend on each other for variable substitution. For example, if one component uses a parameter value from another component for the variable substitution, the product must refresh the other component first. If you do not specify Refresh Order in this case, the dependent component can pick up a value that the current refresh has not updated.

Refresh Order has the following values. No specific order is guaranteed within a level. :

- **Don't Care:** The product refreshes the component as if it has the Middle ordinal.
- **Initialization:** The product refreshes components that are set to Initialization first.
- **First**
- **Early**
- **Middle**
- **Late**
- **Last**
- **Cleanup:** The product refreshes components that are set to Cleanup last.

Modifier

Specifies one of the following options: to modify the discovered results:

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

Modifier Parameters

Defines the parameters for the selected Modifier.

Nearest Neighbor

Specifies whether the product selects the component with the closest file system root (in terms of depth) if a component can nest in multiple components.

Default: Yes

Nest By

Specifies the file system relationship that determines whether the component nests in other components.

Note: If you select This Blueprint Will Not Use Nesting, Nest By is not required:

Child

Defines that the component should nest in any component whose file system root is higher in the file system directory tree.

Child or Sibling

Defines that the component should nest in any component whose file system root is equal to or higher in the file system directory tree.

Direct Child

Defines that the component can nest only in components whose file system root is one level higher in the file system directory tree.

Sibling

Defines that the component should nest in any component with the same file system root.

Nest Only In

Defines 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. Use this option to select a specific nesting match or to restrict nesting to an exact location in 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. Use this option to select a specific nesting match or to restrict nesting to an exact depth in the file system directory tree.

Chapter 5: Customize Blueprints with Nesting Modifiers in Component Grouping Options Tab

A few applications do not have a file or registry indicator to determine their existence. However, you can determine their existence by the output from a command or an entry in a file. The nesting modifiers (plug-in programs) are used in Blueprints to discover such applications.

The topics in this section provide guidelines to modify Blueprints that use nesting modifier to discover child components. For example, Oracle and Microsoft SQL Blueprints.

This section contains the following topics:

Oracle and MSSQL Blueprints with Support for Instance Information

Beginning CA Configuration Automation r12.8 SP01, the following Oracle and MSSQL Blueprints are available to discover the respective instances or databases as components:

- Microsoft SQL 2008 Server Instance v10.* r1.0.0 (Parent Blueprint)

  Microsoft SQL 2008 Server Database v10.* r1.0.0 (Child Blueprint)
- Microsoft SQL 2012 Server Instance v11.* r1.0.0 (Parent Blueprint)

  Microsoft SQL 2012 Server Database v11.* r1.0.0 (Child Blueprint)
- Oracle 10g Database (UNIX) v10.* r1.0.0 (Parent Blueprint)

  Oracle 10g Database Instance (UNIX) v10.* r1.0.0 (Child Blueprint)
- Oracle 11g Database (UNIX) v11.* r1.0.0 (Parent Blueprint)

  Oracle 11g Database Instance (UNIX) v11.* r1.0.0 (Child Blueprint)

Customize Oracle and MSSQL Blueprints with Support for Instance Information: Component Grouping Options Tab

A Modifier that is defined in the **Component Grouping Options** tab uses the Blueprint reference that is listed in the **Nest Only In** field. If you customize an existing Blueprint, ensure to change the reference to this Blueprint in the **Nest only In** field in other Blueprints.

For example:

The parent Blueprint Oracle 11g Database (UNIX) v11.* r1.0.0 is referenced in the Oracle 11g Database Instance (UNIX) v11.* r1.0.0 child blueprint. The modifier **Oracle Instance Nesting** uses the child Blueprint reference to discover the child components.

If you customize an existing Blueprint, ensure that you modify both parent and child Blueprint, and the reference from the child to parent Blueprint.

Customize Oracle and MSSQL Blueprints with Support for Instance Information: Component Parameter as Input to Modifier

The Oracle and MSSQL Blueprints use a component parameter to get the list of instances and databases. The modifier that is defined in the Component Grouping Options tab uses these instances and databases to discover the child components.

For example, the following parent MSSQL Blueprints have **TheDatabasesList** as a component parameter to get the list of databases in the MSSQL server instance.

- Microsoft SQL 2008 Server Instance v10.* r1.0.0
- Microsoft SQL 2012 Server Instance v11.* r1.0.0

The parameter uses a script to query the SQL server through ADODB.Connection objects to get the list of databases.

Similarly, the following parent Oracle Blueprints have **Instances** as a component parameter to discover the child components in the Oracle database.

- Oracle 10g Database (UNIX) v10.* r1.0.0
- Oracle 11g Database (UNIX) v11.* r1.0.0

The parameter uses the following script that parses oratab file entries:

```
list=`cat /etc/oratab | grep "$(Root)" | cut -f 1 -d :`;for inst in $list;do echo
"$inst,";done;
```

To meet your environment's requirement, you can modify the component parameter to discover the list of instances or databases that a modifier uses.

Index

R

S

U

V