CML Tutorial

IN THIS TUTORIAL

This tutorial contains the following sections:

About the CML Tutorial

This guide shows you how to use Opsware's Configuration Markup Language (CML) to make an Opsware Application Configuration Template based upon the Microsoft Internet Information Services (IIS) Web server configuration file UrlScan.ini.

While this tutorial will not teach you everything there possibly is to know about CML, creating a CML template from UrlScan.ini will help you gain both a fundamental understanding of CML and the process of creating an Application Configuration Template from a real configuration file.

For more information on how to use Application Configurations in the OCC Client to manage your applications in your managed server environment, see the *Opsware® SAS User's Guide*.

CML Fundamentals

This section contains the main terms and concepts you will need to be familiar with in order to understand this tutorial.

- What is an Application Configuration Template?
- What is an Application Configuration?
- What is CML?
- About the CML Parser
- Anatomy of a CML Tag
- CML Tags You Should Know

What is an Application Configuration Template?

An application configuration template is a “templatized” version of an actual configuration file whose values have been turned into variables. Using the OCC Client, a user can edit a template’s value sets and propagate those changes to an actual configuration file on a server.

Once the template version of the configuration file has been created and added to an Application Configuration (inside the OCC Client user interface), system administrators can easily define values for configuration files on servers and server groups.

A CML template is a text file that uses the TPL extension. The Application Configuration feature accepts non-ascii characters, but all key names in your CML templates must be in ASCII. Other fields and text can be either ASCII or non ASCII text.

What is an Application Configuration?

An Application Configuration a like a container or folder which houses Application Configuration Templates. If an application contains several configuration files, you can create an Application Configuration Template for each configuration file you want to manage, and then create a single Application Configuration to contain all the templates. In addition to housing Application Configuration Templates, an Application Configuration can also contain Pre and Post-install scripts that can be executed before and after a configuration push.

What is CML?

CML (Configuration Markup Language) is a configuration markup language that allows system administrators to “templatize” or variablize entries in a native configuration file so those files can be edited and managed from a single location inside the OCC Client.

CML uses special markup tags to modify an application’s configuration’s file’s content – data such as directives, definitions, and so on – so that the configuration data becomes transformed into variables. Once the configuration file has been templatized and added to an Application Configuration inside the OCC Client, the end user can manage, edit, and make changes to the native configuration files on managed servers.

About the CML Parser

The CML parser is the engine that utilizes CML configuration files to extract values from existing template files, where they can then be edited. The parser also uses the same CML configuration files to regenerate those configuration files, with new values. For this tutorial, you do not need to know the technical details of the CML parser. You do need to know that CML is used to represent the structure and format of a configuration file, allowing editing of live configuration files on managed servers.

Anatomy of a CML Tag

The basic structure, or anatomy, of a CML tag follows this structure:

```
@<level><tag type><name>;<data
type>;<range>;<option1>;<option2>;(more options)@
```

At its most fundamental level, each CML tag starts and end with the @ symbol. Everything else between consists of one or more of the following CML elements:

- level
- tag type
- name
- data type
- range
- options

level

Levels are used to nest blocks within other blocks. (A block defines parsing rules for a section of information inside a native configuration file.) Levels also signify whether the block spans multiple lines or just one line. Levels under 100 span multiple lines; levels over 100 are contained in a single line. Blocks that start at a level less than or equal to the surrounding block close that block, so when you are nesting blocks use block levels higher than the encompassing block.

You would want to use nested blocks to specify how levels will exist in a block of information in the template. For example, if you had a section in a configuration file that contained two types of data, such as ordered lines and unordered line, then you could set two or more nested blocks (at different levels) in order to handle the information in the section.

tag type

Indicates the type of tag, denoted by a symbol, such as: comment (`#`), loop: (`*`), loop target (`.`), instruction (`!`), and so on.

name

Indicates where the data read by this tag is stored in namespace. Names can be either absolute or relative. For example:

- **Relative**: If you defined `@!namespace="/system/service/webserver/"@`, then the relative name `@ListenPort@` would use namespace /system/service/webserver/ListenPort. Subsequently, you could then us the relative name `@LogFile@` and this name would use the namespace /system/service/webserver/LogFile.
- **Absolute**: For example, `@/system/service/webserver/ListenPort@`.

If you define an absolute namespace in the header of a template, then all relative instances of a name will be appended to that namespace. Conversely, you can define an absolute name any time you wish inside the template, and a new namespace will be created.

data type

Indicates what data type the tag will handle. Basic data types can be string, int, decimal, hostname, ordered-string-set, unordered-int-set, ordered-hostname-list, and so on.

range

Specifies ranges of data. For example, you could use a range specifier to specific data range for an int type1 > 2000, or the range for a string type: “Windows”, “Linux” and so on.

options

Options are very similar to instruction tags and serve to modify or affect the behavior of the tag. They can also be appended to the end most tags, separated by semicolons. For example:

```
;delimeter-is-comma;optional;boolean-yes-format=yes;boolean-no-
format=no;
```

CML Tags You Should Know

In order to create a CML template for UrlScan.ini, you should become familiar with the following CML tags:

- Comment Tag
- Instruction Tag
- Replace Tag
- Block Tag
- Loop Tag
- Loop Target Tag

Comment Tag

The comment tag can be used to insert information about the template, the configuration file it represents, template metadata (creator, applicable systems, etc.) or any other human-readable information.

Syntax

```
@# <one line comment>EOL
```

Or

```
@## <comments spanning multiple lines> #@
```

The comment tag is often used at the beginning of a CML template (the header) so the author can provide information about the template, such as the name of the template, the configuration file the template as based upon, the purpose of the template, a description of the template, the author, the date, and so on.

Instruction Tag

The instruction tag sets options that will be used at parse time. For example, defining the namespace, whether a list is sorted, ordered, or unordered, how the parser should interpret white space, acceptable delimiters, defining comment characters, and so on.

Syntax

```
@![{options}]@
```

Replace Tag

The replace tag functions to replace the tag in a CML line with the data from that location in namespace. It is an indicator that the text in this location is data, and it also specifies details about how that data should be stored and validated.

Syntax

```
@{source}[;[{type}][;[{range}[;{option}[;{option}]...]]]]@
```

The only required element in a replace tag is the source; everything else is optional.

Block Tag

The block tag allows you to group related configuration statements. A block defines parsing rules for a section of information inside a native configuration file. For example, you might have a section of a configuration file where true/false values are defined as either 1/0. In another section in the same file true/false values are set to T/F. You could use the use the block tag to separate the two different ways the CML parser interprets these different ways of defining true/false.

Another example would be if in one section of a configuration file a specific number of spaces are important, while in another section any number of spaces is acceptable, you would use the block tag to indicate where the configuration statements differ.

Syntax

```
@[{level}][[;{option}[;{option}]...]]]]@ <block> <explicit
or implicit block end>
```

Loop Tag

The loop tag allows sequences (lists and sets) to be enumerated. The block associated with a loop element will be processed for each incident of that block in an input file, and will be generated in an output file for each incidence of that data in a valueset.

Syntax

```
@[{level}]*{source}[;[{type}][;[{range}[;{option}[;{option}]
...]]]]@ <block> <explicit or implicit block end>
```

Loop Target Tag

The loop target tag is used in a block that is encapsulated by a sequence or type other than namespace. When encountered in a block, this tag is simply replaced with the value of the current value with each loop iteration.

Syntax

```
@.@
```

Creating a CML Template

This section shows you how to create an Application Configuration using CML and contains the following sections:

- Materials Needed for the Tutorial
- Completed Template Sample
- 1. Familiarize Yourself with the Native Configuration File and Its Documentation
- 2. Create CML Template File for UrlScan.ini
- 3. Create the CML Template Header
- 4. Create the CML Template Basic Setup Section
- 5. Create the Template Body
- 6. Mark Up UrlScan [Options] Section – Opening Blocks
- 7. Closing One Block by Opening a New One – Marking Up [AllowExtensions]
- 8. Mark Up [DenyExtensions] Section by Opening a New Block
- 9. Mark Up [AllowVerbs] and [DenyVerbs] Sections
- 10. Mark Up [DenyHeaders] Section
- 11. Mark Up [DenyURLSequences] Section
- 12. Mark Up [RequestLimits] Section
- 13. From Template to Application Configuration
- Completed Template Sample

Materials Needed for the Tutorial

- Documentation for UrlScan.ini
- UrlScan.ini file
- A text editor

Completed Template Sample

After you finish this tutorial, you can look at the completed url_scan_ini.tpl template so you can compare your results with a completed template. To view the completed template, see “Completed url_scan_ini.tpl Template” on page 30.

1. Familiarize Yourself with the Native Configuration File and Its Documentation

Once you have identified an application configuration file you want to manage with ACM, the first thing to is to analyze the native configuration file and its documentation. Make sure that you understand the purpose of the configuration file and all the elements

For example, the documentation for UrlScan.ini tells you that the configuration file enables systems administrators to configure IIS to screen and analyze HTTP requests in order to prevent Internet attacks.

UrlScan.ini consists of several sections, such as [Options], [AllowVerbs], [DenyVerbs], [DenyHeaders], [AllowExtensions], and [DenyExtensions]. Each section allows you to set different configurations to either allow or not allow certain kinds of HTTP requests on your IIS Server.

Each of these sections, judging from the documenting, do not need to be arranged in any specific order. For example, I could list the [Options] sections followed by [DenyVerbs] instead of [AllowVerbs], and the file would still contain the same configuration information and perform its function within IIS. In other words, the order of the main sections of the configuration file is not important

However, the information inside each of these sections do need to be listed (ordered) in a specific way. In other words, the [AllowVerbs] section must be followed by specific verbs you do not want to allow to access your web site. For example, if you put the actual verbs before the [AllowVerbs] string, then that feature of the configuration file would not work.

You also want to become familiar with the kinds of data the configuration file manages. In general, UrlScan.ini works with lists of strings, such as lists of verbs and file extensions. In addition, the file also allows the user to set several yes or no (boolean) options. This kind of information is useful to know before you start creating an Application Configuration Template.

2. Create CML Template File for UrlScan.ini

A CML Template begins as a simple text file that uses the TPL extension.

To create the UrlScan.ini template:

1. Using a text editor, create a new text file and save it as Url_Scan_ini.tpl. TPL is the file extension used by Opsware for CML templates, though technically you can use any file extension you want, or none at all. Opsware Application Configuration Template file naming conventions typically uses the name of the native configuration file with underscores between each section of the native configuration filename.
2. Now that you have created the CML template file, you are now ready to build the basic structure of the template, which will consist of a Header, Basic Setup Section, and Template Body.

3. Create the CML Template Header

The purpose of creating the CML template header is so that anyone who reads this template will know:

- Name of the native file this template manages (file's absolute pathname)
- Operating systems that the file can work on
- Version of the template
- Author of the template (optional: author's email address)

The first CML tag you will use to create the template's header will be the Comment tag, which allows you to write information about the template. The Comment tag uses this syntax:

```
@# <one line comment>EOL
```

Or

```
@## <comments spanning multiple lines> #@
```

To create the CML template header, using the CML Comment tag, create a header section at the top of the file that contains three lines of content, the native configuration file that the template will manage, the template version, and the author (with email address).

For example, here's what your template header might look like:

```
@##############################################
# #
# \system32\inetsrv\urlscan.ini (Windows) #
# Version 1.0 #
# Joe Author (joe_author@your_company.com) #
# #
##############################################@
```

4. Create the CML Template Basic Setup Section

This basic setup section is where you list CML options that instruct the parser how to interpret the CML file. This section can include such as namespace definition, white space handling, list rules, line rules, and so on.

To create the CML template basic setup section:

1. Following the header of the CML template, enter the following information (or copy and paste from here):

   ```
   @!namespace=/security/@
   @!filename-key="/test";filename-default="/c/UrlScan.ini"@
   @!optional-whitespace@
   @!boolean-yes-format="1";boolean-no-format="0"@
   @!line-comment-is-semicolon@
   @!unordered-lines@
   ```

   This information defines important rules for the url_scan_ini.tpl template, indicating how the CML parser is supposed to interpret and handle information in the template. Notice that each line is a CML instruction tag. You know this is a CML instruction tag because the way the tag starts:

   `@!`

   with an at (`@`) sign and an exclamation mark (`!`).

CML Template Basic Setup Section Explained

Table 1 explains what each section of the template basic setup section means and does.

Table 1: CML Template Basic Setup Section Explained

CML TAG	DESCRIPTION
`@!namespace=/security/@`	Define the namespace; in other words, this defines where in the Opsware Model Repository values read by the CML template will be stored.

`@!filename-key="/files/urlscan_ini";filename-default="/c/urlscan.ini"@`	`filename-key`
	Defines the location in namespace where the filename will stored.
	
	`filename-default`
	Defines the location where the native configuration file will be saved on the disk. This path can be changed by the user from the OCC Client.
	
	Note that the path names use only forward slashes.

`@!optional-whitespace@`	Indicates that whitespace is optional between items in the configuration file. For example, either of the following entries would be valid if this option is set:
	
	`Key = "value"`
	
	`Key="value"`

`@!boolean-yes-format="1";boolean-no-format="0"@`	Defines the allowable boolean values in the configuration file. In this case, Yes is indicated with the character `1`, and No is indicated with a `0`. This means that is a user tried to use the string `yes`, the Application Configuration would not accept it.

Table 1: CML Template Basic Setup Section Explained

CML TAG	DESCRIPTION
`@!line-comment-is-semicolon@`	Instructs the parser not to read anything that follows a semicolon in the configuration file. This allows an end user to make comments in the native configuration file using the semicolon before each comment.
`@!unordered-lines@`	Tells the parser that the sections in the configuration file can be in any order. If you used `ordered-lines`, then the configuration file would have to conform to the order of the template.

5. Create the Template Body

Now that you have created both the header and basic setup portions of your CML template, you are now ready to construct the body. The body is where all your main instructions will be contained.

To create the template body:

1. The first thing to do is create a heading that indicates to anyone who might read this file that this is the beginning of the body of the template. Enter the following at the end of the basic setup section of the template:

```
@############################################
# Begin data                                #
############################################@
```

2. Save the changes to the file.

6. Mark Up UrlScan [Options] Section – Opening Blocks

Now you are ready to start marking up the template. The first section of the UrlScan.ini file you will convert into CML is the [Options] section, which contains several options for the configuration file.

In CML, if a section of information in a configuration file has more than one kind of data (data that needs to be read differently by the CML parser), you can open “blocks” to handle each section of information separately. Typically, you open a block in CML in order to define special parser rules for a section of the CML file. In the case of the [Options] section, there are basically two “blocks” of information that need to be read by the CML parser: the title of the section and all the options. Since both of these blocks belong together, you will set them at different levels, the first block (the title of the section) at level one, and the second block (the contents of the section) at level two. Nesting the blocks in this manner keeps the sections within the block together when read by the parser.

To markup the UrlScan.ini [Options] section:

1. After the “begin data” section of the template, enter the following:

   ```
   @1[;optional;ordered-lines@
   [Options]
   @2[;unordered-lines@
   ```

2. In the UrlScan.ini file the [Options] section contains a list of key value pairs. We will use the block tag (`[`) set at two levels because there are two kinds of data in this section: a heading and followed by a list of key value pairs. The first level block handles the text string “[Options]” while the second level block will handle all of the key value pairs in that section.

Table 2 explains how to open two block levels for the [Options] section.

Table 2: Marking Up the Start of the [Options] Section

CML TAG	DESCRIPTION

`@1[;optional;ordered-lines@`	The number 1 sets the first level of the multiline block.
	
	`[`
	CML block symbol opens a new block.
	
	`optional`
	Indicates that this entire block is optional and not required to be in the configuration file for the file to be "correct".
	
	`ordered-lines`
	Indicates that whatever follows this tag (the string [Options]) has to come first in the native UrlScan.ini configuration file. In other words, you could not list in the native file all the options and then the title. “[Options]” has to come first. In CML, the option “`ordered-lines`” determines this order.

`[Options]`	The string that names the section in the native configuration file.

`@2[;unordered-lines@`	The number 2 sets the second level of the block.
	
	`[`
	CML block symbol opens a new block.
	
	`unordered lines`
	Indicates that all the lines that follow [Options] within the block can be in any order in the configuration file. In other words, all the key value pairs that are contained in the [Options] section can be ordered and will be read by parser.

3 Next, you will markup all the options lines from the configuration file. Most of these entries use the CML replace tag because they are simply key value pairs that allow a user to replace a single value. Table 3 explains the CML markup of each option.

Table 3: Marked Up Key Value Pairs from UrlScan.ini [Options] Section

CML TAG	DESCRIPTION

`AllowDotInPath = @allow_dot_in_path;boolean@`	Note: All of the key value pair markup use some variation of the following syntax (unless otherwise indicated):
	
	`string literal = @source;type@`
	
	`allow_dot_in_path`
	This string defines the namespace path to store this value. In this example, the namespace is relative, which means that it will be appended to the namespace that you defined in the header of the template (@!namespace=/security/@) and will store the value in that namespace location.
	
	For example:
	`/security/allow_dot_in_path.`
	
	If you wanted, you could also write this tag like this:
	`AllowDotInPath = @/security/allow_dot_in_path;boolean@`
	
	`boolean`
	
	Since the key value pair type is boolean, we used the CML type: `boolean`. Note that since in the header of this template we defined an acceptable boolean yes value as 1, when the end user modifies the template in the OCC Client, they would need to enter a one if they want to allow dots in the path of IIS.

Table 3: Marked Up Key Value Pairs from UrlScan.ini [Options] Section

CML TAG	DESCRIPTION
`AllowHighBitCharacters = @allow_high_bit_characters;boolean@`	Allows users to choose whether or not high bit characters are acceptable in a URL, flagged by a yes (1) or no (2) in the configuration file.
`AllowLateScanning = @allow_late_scanning;boolean@`	Allows users to choose whether or not late scanning of a URL is acceptable. And, defines a namespace location to store value. `boolean` indicates this key is accepts a yes (1) or no (2) in the configuration file.
`AlternateServerName = @alternate_servername@`	Defines a namespace where an alternate server name can be stored when entered by the user, or read in from a configuration file.
`EnableLogging = @enable_logging;boolean@`	Allows users to turn on logging, flagged by a yes (1) or no (2) in the configuration file.
`LoggingDirectory = @logging_directory;dir@`	Allows users to choose a directory to store log files, if logging has been turned on. Notice that for the type, the CML tag uses the element `dir` - an acceptable CML data type.
`LogLongURLs = @log_long_urls;boolean@`	Allows user to choose whether or not to log URLs that access the server, a yes (1) or no (2) in the configuration file.
`NormalizeUrlBeforeScan = @normalize_url_before_scan;boolean@`	Allows users to choose whether or not to normalize the URL before it is read by the server, flagged by a yes (1) or no (2) in the configuration file.
`PerDayLogging = @per_day_logging;boolean@`	Allows users to choose to turn on per day logging, flagged by a yes (1) or no (2) in the configuration file.

Table 3: Marked Up Key Value Pairs from UrlScan.ini [Options] Section

CML TAG	DESCRIPTION
`PerProcessLogging = @per_process_logging;boolean@`	Allows users to turn on or off per process logging, flagged by a yes (1) or no (2) in the configuration file.

`RejectResponseUrl =`
 	
`@reject_response_url;string;r'(HTTP_URLSCAN_STATUS_HEADER)|(HTTP_URLSCAN_ORIGINAL_VERB)|(HTTP_URLSCAN_ORIGINAL_URL)';optional@`
	Syntax
	`string literal = @source;type;r'regular expression';option@`
	
	`reject response`
	String literal that defines the path where the strings will be stored in namespace.
	
	`string`
	Indicates that the data type for the reject URL request is a string.
	
	`r’`
	A string range specifier that introduces a regular expression. In this case, a range of string literals.
	
	`(HTTP_URLSCAN_STATUS_HEADER)|(HTTP_URLSCAN_ORIGINAL_VERB)|(HTTP_URLSCAN_ORIGINAL_URL)'`
	The string literals (rejected URL responses) to be read by the parser: the status header, original verb, and original URL.
	
	`optional`
	Indicates that this value is optional. That is, if left blank, the parser can still read the CML.

Table 3: Marked Up Key Value Pairs from UrlScan.ini [Options] Section

CML TAG	DESCRIPTION
`RemoveServerHeader = @remove_server_header;boolean@`	Allows users to turn on or off the RemoveServerHeading feature. When activated (set to 1), the reject response sent to the client will removing the server header in the message. This setting is flagged by a yes (1) or no (2) in the configuration file.
`UseAllowVerbs = @use_allow_verbs;boolean@`	Allows users to turn on or off the UseAllowVerbs feature. When activated (set to 1), the server will reject any request to the server that contain an HTTP verb that is not explicitly listed in the AllowVerbs section of the UrlScan.ini file. Flagged by a yes (1) or no (2) in the configuration file.
`UseAllowExtensions = @use_allow_extensions;boolean@`	Allows users to turn on or off the UseAllowExtension feature. When activated (set to 1), the server will reject any request to the server that contain a file extension that it not explicitly listed in the AllowExtension section of the UrlScan.ini file. Flagged by a yes (1) or no (2) in the configuration file.
`UseFastPathReject = @use_fast_path_reject;boolean@`	Allows users to turn on or off the UseFastPathReject feature. When activated (set to 1), the server ignores the RejectResponseUrl option and returns a short 404 response to the client when a URL is rejected. Flagged by a yes (1) or no (2) in the configuration file.
`VerifyNormalization = @verify_normalization;boolean@`	Allows user to turn on or off normalization of all URLs scanned by UrlScan.ini. When activated (set to 1), the URL is normalized before being scanned. Flagged by a yes (1) or no (2) in the configuration file.

7. Closing One Block by Opening a New One – Marking Up [AllowExtensions]

Now that you have marked up all of the options in the [Options] section of the UrlScan.ini file, you are ready to start marking up the next section, [AllowExtensions]. Remember that to start the [Options] section you had to open a two level block to account for two levels of information – the title of the [Options] section and its contents.

Before you can start marking up the [AllowExtensions], you need to close the previous section by closing the CML block. With CML, you can close a block by opening a new block at a higher (lower number) or equal to level. In this task, you will open the new block for the [AllowExtensions] the same way you opened a block for the [Options] section, by starting a new first level block.

To open a new block and mark up the [AllowExtensions] section:

1. After the last line of the [Options] section, enter the following text to open the new block for the [AllowExtensions] section:

```
@1[;optional;ordered-lines@
[AllowExtensions]
@2[;unordered-lines@
```

Table 4 explains how opening a new two level block closes the previous block.

Table 4: Starting a New Block for the [AllowExtensions] Section

CML TAG	DESCRIPTION

`@1[;optional;ordered-lines@`	The number 1 opens a new level one block. Because it is a number 1 level block, which is at a higher level than the previous block (a level two block for the key value pairs in the [Options] section) and equal to the level 1 block before that, it will close the two blocks that came before it.
	
	Note that you could also close a block by using the close block command. For example:
	`@2]@`
	
	`[`
	CML block symbol that opens a new block.
	
	`optional`
	Indicates that this entire block is optional and not required to be in the configuration file for the file to be "correct".
	
	`ordered-lines`
	Indicates that whatever follows this tag (the string [AllowExtensions] has to come first in the native UrlScan.ini configuration file. In other words, you could not list all the options in the native file and then the title. [AllowExtensions] has to come first. In CML, the ordered-line element determines this order.

`[Options]`	The literal string that names the section in the native configuration file.

Table 4: Starting a New Block for the [AllowExtensions] Section

CML TAG	DESCRIPTION

`@2[;unordered-lines@`	The number 2 sets the second level of the block.
	`[`
	CML block symbol that opens a new block.
	`unordered lines`
	Indicates that all the lines that follow [AllowExtensions] within the block can be in any order in the configuration file. In other words, all the key value pairs that are contained in the [AllowExtensions] section can be ordered in any order you wish.

2 Next, because the [AllowExtensions] section of the UrlScan.ini file can contain any list of file extensions entered by the user, you will use a CML loop and loop target tag to instruct the parser will read the information in this section one line at a time, then repeat by reading the next line, and so on.
Directly after the last `@2[;unordered-lines@` text from the last step, enter the following text:

```
@*allow_extension;unordered-string-set@
.@.@
```

Table 5 explains the how the loop and loop target CML tags work:

Table 5: Loop and Loop Target CML Tags

CML TAG	DESCRIPTION

`@*allow_extension;unordered-string-set@`	**Syntax**
	`@<level><tag type><name>;<data type>;<options>@`
	The loop tag (`*`) will "loop" or read over the unordered string set listed in the [AllowExtensions] section.
	`allow_extension`
	String that defines the path where the strings will be stored in namespace.
	`unordered-string-set`
	Indicates that the list of strings do not have to be listed in any specific order.

`.@.@`	First (`.`) In this section, this unordered string set that the parser reads is a list of file extensions listed in the [AllowExtensions] section that start with a (`.`) character. `@.@` Loop target tag (`.`) instructs the parser to read everything in this list that starts with a period character.

3 Save the file.

8. Mark Up [DenyExtensions] Section by Opening a New Block

In this task, you will markup the [DenyExtensions] section of the UrlScan.ini file the exact same way you marked up the [AllowExtensions] section. You will be opening a new level one block, which closes the previously opened block from the [AllowExtensions] section.

Then, you will open a level two block from which you will instruct the parser to read an unordered list of all file extensions beginning with a (.) that you wish to block using UrlScan.ini.

The CML markup for the [AllowExtensions] section looks like this:

```
@1[;optional;ordered-lines@
[DenyExtensions]
@2[;unordered-lines@
@*deny_extension;unordered-string-set@
.@.@
```

9. Mark Up [AllowVerbs] and [DenyVerbs] Sections

The next two sections of the UrlScan.ini file will follow the exact same CML markup as you used for [DenyExtensions] in the previous sections. You will open a first level block to close the previous block, which will also parse the following text as an ordered line.

Then, you will open a second level block that reads the following list of as an unordered strings – in other words, a list of verbs. In these two sections, the string you will instruct CML to read will be a list of verbs you wish to allow into your web site and a list of verbs you wish to deny access to your web site.

The CML markup for both of these sections is as follows:

```
@1[;optional;ordered-lines@
[AllowVerbs]
@2[;unordered-lines@
@*allow_verb;unordered-string-set@
@.@

@1[;optional;ordered-lines@
[DenyVerbs]
@2[;unordered-lines@
@*deny_verb;unordered-string-set@
@.@
```

10. Mark Up [DenyHeaders] Section

In this next task, you will mark up the [DenyHeaders] section of the UrlScan.ini file, which allows you to configure IIS to deny specific HTTP request headers.

This section will be marked up in CML similarly to the previous sections in that you will open two blocks that will be read for strings. However, you will be separating the list of HTTP headers listed in the UrlScan.ini file by a colon, using a CML sequence delimiter. Since HTTP request headers contain a colon (:), you need to use a sequence delimiter to tell the parser to read each line in the section so when it encounters a colon (:), it will move on to the next entry.

For example, the list of HTTP headers to be denied listed in the UrlScan.ini file might read something like this:

Translate:
If:
Lock-Token:

Because each header request listed in the configuration file ends with a (:), we need to instruct the parser to recognize the (:) as the end of an entry.

To markup the [DenyHeaders] section:

1. After the last line of the [DenyVerbs] section, enter the following text to open the new block for the [DenyHeaders] section:

   ```
   @1[;optional;ordered-lines@
   [DenyHeaders]
   @2[;unordered-lines@
   ```

   As you have done in previous sections, with these tags you are opening a level one block to be read as an ordered line, then opening a second level block to be read as unordered lines.

2. Next, type the following CML loop and loop target tags to instruct the parser to read through the list of header requests:

   ```
   @*deny_header;unordered-string-set;;sequence-delimiter=":"@
   @.@:
   ```

Loop and Loop Target Tags for the [DenyHeaders] SectionTable 6 describes the syntax of these two tags.

Table 6: Loop and Loop Target Tags for the [DenyHeaders] Section

CML TAG	DESCRIPTION

`@*deny_header;unordered-string-set;;sequence-delimiter=":"@`	`*`
	Indicates a loop CML tag that will read through the list of strings.
	
	`deny_header`
	String literal that defines the path where the strings will be stored in namespace.
	
	`unordered-string-set`
	Indicates that the list of strings can be listed in any order.
	
	`;`
	The first semicolon separates the two sections of the tag.
	
	`;`
	The second semicolon allows you to enter the following colon (:) sequence delimeter without it being interpreted as a range.
	
	`sequence-delimiter=":"`
	Instructs the parser to read a colon (:) as part of the string and the point at which to move on to the next entry.

`@.@`	Loop target tag instructs the parser to store these values into the `deny_header` namespace. E.g., /security/deny_extension
`:`	Final colon (:) tells the parser that each item in this list is going to be followed by a colon. In other words, this character will be included and stored as a part of the entry for a denied header.

3 Save the file.

11. Mark Up [DenyURLSequences] Section

Marking up the [DenyUrlSequence] is very similar to the way in which you marked up the [DenyHeader] section: you will open two blocks that will be read for order and unordered strings. However, for this section you will be separating the list of URL sequences in the template with a field delimiter. The field delimiter used here will be an end of line element (eol) which instructs the parser stop reading an entry when it encounters the end of a line.

To markup the [DenyUrlSequence] section:

1 After the last line of the [DenyUrlSequence] section, enter the following text to open the new block for the [DenyUrlSequence] section:

```
@1[;optional;ordered-lines@
[DenyUrlSequence]
@2[;unordered-lines@
```

As you have done in previous sections, with these tags you are opening a level one block to be read as an ordered line, then opening a second level block to be read as unordered lines.

2 Next, type the following CML loop and loop target tags to instruct the parser to read through the list of URL sequences to be denied:

```
@*deny_url_sequence;unordered-string-set;;field-delimiter-
is-eol@
@.@
```

Table 7 describes the syntax of these tags

Table 7: Loop and Loop Target Tags for the [DenyUrlSequence] Section

CML TAG	DESCRIPTION

`@*deny_url_sequence;unordered-string-set;;field-delimiter-is-eol@`	`*`
	Indicates a loop CML tag that will read through the list of strings.
	
	`deny_url_sequence`
	String literal that defines the path where the string will be stored in namespace.
	
	`unordered-string-set`
	Indicates that the list of strings can be listed in any order.
	
	`;`
	The first semicolon separates the two sections of the tag.
	
	`;`
	The second semicolon allows you to enter the following colon (:) sequence delimeter without it being interpreted as a range.
	
	`sequence-delimiter=":"`
	Instructs the parser to read a colon (:) as part of the string and the point at which to move on to the next entry.

`@.@`	Loop target tag instructs the parser to store these values into the `deny_url_sequence` namespace. E.g., /security/deny_url_sequence.

3 Save the file.

12. Mark Up [RequestLimits] Section

Marking up the [RequestsLimits] is very similar to the way in which you marked up the [DenyUrlSequence] section: you will open two blocks that will be read for order and unordered strings. But for this section, after you open both blocks, you will be using the CML replace tag to mark up three key value pairs.

To markup the [RequestsLimits] section:

1. After the last line of the [RequestsLimits] section, enter the following text to open the new block for the [RequestsLimits] section:

```
@1[;optional;ordered-lines@
[RequestsLimits]
@2[;unordered-lines@
```

   As you have done in previous sections, with these tags you are opening a level one block to be read as an ordered line, then opening a second level block to be read as unordered lines. Recall that by starting the new first level block, you are closing the previous second level block from the {DenyUrlSequence] section.

2. Next, type the following CML replace tags to mark up the three kay value pairs found in the [RequestLimits] section:

```
MaxAllowedContentLength = @max_allowed_content_length;int@
MaxUrl = @max_url;int@
MaxQueryString = @max_query_string;int@
@1]@
```

Table 8 describes the syntax of these tags.

Table 8: Loop and Loop Target Tags for the [DenyUrlSequence] Section

CML TAG	DESCRIPTION
`MaxAllowedContentLength = @max_allowed_content_length;int@`	`MaxAllowedContentLength` Request limit parameter string from the configuration file. `max_allowed_content_length` String literal that defines the path where the value will be stored in namespace. `int` Indicates that the value to be stored is an integer.
`MaxUrl = @max_url;int@`	`MaxUrl` Request limit parameter string from the configuration file. `max_url` String literal that defines the path where the value will be stored in namespace. `int` Indicates that the value to be stored is an integer.
`MaxQueryString = @max_query_string;int@`	`MaxQueryString` Request limit parameter string from the configuration file. `max_query_string` String literal that defines the path where the value will be stored in namespace. `int` Indicates that the value to be stored is an integer.
`@1]@`	This level one block tag closes the block.

3 Save the File

13. From Template to Application Configuration

Once you have completed creating the CML template for UrlScan.ini (saved as url_scan_ ini.tpl), you are now ready to do the following tasks:

- Import the template into the OCC Client
- Add the template to an Application Configuration
- Validate the CML syntax
- Attach the Application Configuration to a server
- Test by making changes and pushing changes to the server

For information on how to create an Application Configuration, add a template to it, validate its CML syntax, and attach it to a server, and push changes, see the online help for Application Configuration in the OCC Client.

Completed url_scan_ini.tpl Template

We have includes a sample of a completed url_Scan_ini.tpl template so you can compare you work with a finished template.

```
@#############################################
# #
# \system32\inetsrv\urlscan.ini (Windows) #
# Version 1.0 #
# Joe Author (joe_author@your_company.com) #
# #
#############################################@

@!namespace=/security/@
@!filename-key="/test";filename-default="/c/UrlScan.ini"@
@!optional-whitespace@
@!boolean-yes-format="1";boolean-no-format="0"@
@!line-comment-is-semicolon@
@!unordered-lines@


@###########################################
# Begin data                              #
###########################################@
```

```
@1[;optional;ordered-lines@
[Options]
@2[;unordered-lines@

AllowDotInPath = @allow_dot_in_path;boolean@

AllowHighBitCharacters = @allow_high_bit_characters;boolean@

AllowLateScanning = @allow_late_scanning;boolean@

AlternateServerName = @alternate_servername@

EnableLogging = @enable_logging;boolean@

LoggingDirectory = @logging_directory;dir@

LogLongURLs = @log_long_urls;boolean@

NormalizeUrlBeforeScan = @normalize_url_before_scan;boolean@

PerDayLogging = @per_day_logging;boolean@

PerProcessLogging = @per_process_logging;boolean@

RejectResponseUrl =
@reject_response_url;string;r'(HTTP_URLSCAN_STATUS_
HEADER)|(HTTP_URLSCAN
_ORIGINAL_VERB)|(HTTP_URLSCAN_ORIGINAL_URL)';optional@

RemoveServerHeader = @remove_server_header;boolean@

UseAllowVerbs = @use_allow_verbs;boolean@

UseAllowExtensions = @use_allow_extensions;boolean@

UseFastPathReject = @use_fast_path_reject;boolean@

VerifyNormalization = @verify_normalization;boolean@

@1[;optional;ordered-lines@
[AllowExtensions]
@2[;unordered-lines@

@*allow_extension;unordered-string-set@
.@.@
```

```
@1[;optional;ordered-lines@
[DenyExtensions]
@2[;unordered-lines@

@*deny_extension;unordered-string-set@
.@.@

@1[;optional;ordered-lines@
[AllowVerbs]
@2[;unordered-lines@

@*allow_verb;unordered-string-set@
@.@

@1[;optional;ordered-lines@
[DenyVerbs]
@2[;unordered-lines@

@*deny_verb;unordered-string-set@
@.@

@1[;optional;ordered-lines@
[DenyHeaders]
@2[;unordered-lines@

@*deny_header;unordered-string-set;;sequence-delimiter=":"@
@.@:

@1[;optional;ordered-lines@
[DenyURLSequences]
@2[;unordered-lines@

@*deny_url_sequence;unordered-string-set;;field-delimiter-is-
eol@
@.@

@1[;optional;ordered-lines@
[RequestLimits]
@2[;unordered-lines@

MaxAllowedContentLength = @max_allowed_content_length;int@

MaxUrl = @max_url;int@

MaxQueryString = @max_query_string;int@
@1]@
```

Using DTD Tags in CML

CML supports Document Type Definition (DTD) tags that can be used to pre-define attributes for a CML tag. Using a DTD tag in CML allows you to change some aspects of how the template is displayed in the OCC Client. The DTD definition generally goes in the beginning of a file and the tag gets shortened to just a name and a tag type.

The main advantage of using DTD tags in CML is the ability to define 'printable' and 'description' values, which are reflected in the OCC client, improving usability. DTD definitions can be used to define any tag that has a name; for example loop tags, loop target tags, replace tags, and so on, but not tags like instruction tags or block tags. DTD tags in CML are also inherently multi-line tags.

DTD Tags Example

Here we will take a tag and create a DTD version of that tag. A DTD tag in CML isn't much different than a regular CML tag; it contains all the elements of a tag minus the "tag type".

For example, in the CML tag below:

```
@*deny_header;unordered-string-set;;sequence-
delimiter=":";optional@
```

this is an instance representing the following format in CML:

```
@<tag type><name>;<data type>;;<option1>;<option2>@
```

The DTD version of this takes the existing elements and reorders them as follows:

```
<start code block>
@~<name>
type = <data type>
description = <description>
printable = <printable>
<option1>
<option2>
...
@
@<tag type><name>@
<end code block>
```

As you can see, this usage also allows for the addition of two new elements: "description" and "printable". Defining "printable" will define the main text for this tag in the OCC Client. Defining "description" will create a description for this value in the OCC Client that is viewable when the user mouses over the field in the Value Set Editor in the OCC Client.

Here is the same tag in full DTD format:

```
<start code block>
@~deny_header
type = unordered-string-set
printable = Headers to Deny
description = This is a list of headers that IIS should deny
sequence-delimiter = ":"
optional
@
@*deny_header@
<end code block>
```

There are a couple things to notice in the example above. In defining a value for “description,” the value can span multiple lines, as long as the lines following the first line have whitespace as the first character.

Options go on a line by themselves, where you have `<option>=<value>` you need to insert spaces before and after the `"="` sign.

Now, where ever you use the tag `@*deny_header@`, the parser will use the predefined DTD for all that tags' information.

Redefining a DTD defined tag, `@*deny_header@`, by using a line like `@*deny_header;unordered-string-set@` will cause the CML template to become invalid.

Note also that DTD style CML is not currently required, but is most obvious when viewing the Application Configuration the OCC Client. If you don't use DTD tags you will not see the 'printable' and 'description' fields, instead you will only see the underlying variable name.

Sequence Aggregation

Because Application Configuration values can be set across many different levels in the Application Configuration inheritance hierarchy (also referred to as the inheritance scope), it is important that you be able control the way multiple sequence values are merged together when you push an Application Configuration on to a server.

ACM allows you to control the way sequence values are merged across inheritance scopes. This means that you can, for example, add some values to a sequence in the Customer scope, Group scope, and the Server scope, and all the values will be merged together to form the final sequence.

The manner in which sequence values are merged is controlled by special tags in the CML template, using three different sequence merge modes:

- **Sequence Replace**: Sequence values from more specific scopes completely replace those from less specific scopes. This occurs for both sequences of sets and lists.
- **Sequence Append**: For lists, values at more general scopes are appended (placed after) to those at more specific scopes. Duplicates, if present, are not removed. For sets, the behavior is the same, except duplicates are merged. For lists, duplicates are identified according to child elements marked with the `primary-key` tag, and then merged. For scalars, this is done by simply removing duplicate values, leaving only the value from the most specific scope (the last occurrence is the merged sequence). This is the default mode, and will be used if nothing else is specified.
- **Sequence Prepend**: Works the same as append, but values at more general scopes are preprended (placed before) to those at more specific scopes.

For example, with these two sets:

- “a, b” – At a more specific (inner) level of the inheritance scope, for example, server instance level.
- “c, d” – At a more general (outer) of the inheritance scope, for example, the server group level.

When the application configuration template is pushed onto the server, the merging results would be:

- Sequence replace: “a, b”
- Sequence append: “a, b, c, d”
- Sequence prepend: “c, d, a, b”

Sequence aggregation occurs not only between scopes, but also within a scope itself. This is evident if there are duplicate values within a sequence of namespaces.

Sequence Replace

In the Replace merge mode (CML tag “`sequence-replace`”), the contents of a sequence defined at a particular scope replace those of less specific scopes, and no merging is performed on the individual elements of the sequence.

For example, if the `sequence-replace` tag has been set for a list in an Application Configuration Template CML source, then values set for that list at the server instance level will override, or replace, those set at the group level and at the Application Configuration default values level.

For example, if a list in an `etc/hosts` file was defined at the group level (outer) as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine
/system/dns/host/2/ip           10.10.10.10
/system/dns/host/2/hostnames/1  loghost
```

And the same list was defined at the device scope (inner), as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine.mydomain.net
/system/dns/host/2/ip           10.10.10.100
/system/dns/host/2/hostnames/1  mailserver
```

If template had defined the `/system/dns/host` element with the `sequence-replace` tag, the final results of the configuration file on the server after the push would be:

```
127.0.0.1 localhost mymachine.mydomain.net
10.10.10.100 mailserver
```

Sequence Append

When the append list merge mode (CML tag “`sequence-append`”) is used for sequences, the values at more general scopes are appended (placed after) those of more specific scopes. Sequence append mode is the default mode for merging list values. If nothing is specified in the CML of the template, the sequence append will be used.

If a list in an `etc/hosts` file was defined at the group level (outer) as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
```

```
/system/dns/host/1/hostnames/2  mymachine
/system/dns/host/2/ip           10.10.10.10
/system/dns/host/2/hostnames/1  loghost
```

And the same list was defined at the device scope (inner), as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine.mydomain.net
/system/dns/host/2/ip           10.10.10.100
/system/dns/host/2/hostnames/1  mailserver
```

Using the value sets from the above example, if the `/system/dns/host` element was a list with the `sequence-append` tag set in the Application Configuration Template, the final results of the configuration file on the server after the push would be:

```
127.0.0.1 localhost mymachine.mydomain.net
10.10.10.100 mailserver
127.0.0.1 localhost mymachine
10.10.10.10 loghost
```

But since it is not allowable for a hosts file to contain duplicate entries, the `/system/dns/host` element will have to be flagged in the Application Configuration Template as a set rather than a list, because sets do not allow duplicates. To avoid duplication of the list values in the example, the Application Configuration Template author would use the Primary Key option.

Primary Key Option in Sequence Merging

When operating in append mode on sets, new values in more specific scopes are appended to those of less specific ones, and duplicate values are merged with the resulting value placed in the resulting sequence according to its position in the more specific scope.

How this affects merged sequence values depends on what kind of data is contained in the sequence:

- For elements in a sequence which are scalars, the value from the most specific scope is used. In other words, values at the server instance level would replace the values at the group level.
- For elements which are namespace sequences, the value is obtained by applying the merge mode specified for that element (in this example, append) based upon matching up the primary fields.

To avoid the duplication of the `/system/dns/host/.ip` value, the Application Configuration Template author would use the CML `primary-key` option. With this option set, ACM will treat entries with the same value for `/system/dns/host/.ip` as the same and merge their contents.

In the example above, the final results of the configuration file on the server after the push would be:

```
127.0.0.1 localhost mymachine.mydomain.net mymachine
10.10.10.100 mailserver
10.10.10.10 loghost
```

Since it is possible to have a set without primary keys, if there are scalars in the sequence, then an aggregation of all scalar values will be used as the primary key. If there are no scalars, then the aggregation of all values in the first sequence will be used as the primary key. Although this is an estimate, in most cases the values will be merged effectively. To ensure that the correct values are used as primary keys, we recommend that you always explicitly set the primary key in a sequence.

Sequence Prepend

When the append list merge mode (CML tag "`sequence-prepend`") is used for sequences, the values at more general scopes are prepended (placed before) those those of more specific scopes.

For example, if a sequence in an `etc/hosts` file was defined at the group level (outer) as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine
/system/dns/host/2/ip           10.10.10.10
/system/dns/host/2/hostnames/1  loghost
```

And the same sequence was defined at the device scope (inner), as the following:

```
/system/dns/host/1/ip           127.0.0.1
/system/dns/host/1/hostnames/1  localhost
/system/dns/host/1/hostnames/2  mymachine.mydomain.net
/system/dns/host/2/ip           10.10.10.100
/system/dns/host/2/hostnames/1  mailserver
```

If the `/system/dns/host` element was a set with the `sequence-prepend` tag set in the Application Configuration Template, the final results of the configuration file on the server after the push would be:

```
10.10.10.10 loghost
127.0.0.1 mymachine localhost mymachine.mydomain.net
10.10.10.100 mailserver
```

CML Grammar

Table 9 describes CML grammar.

Some elements list in the grammar are not covered in the tutorial.

Table 9: CML Grammar

CML TAG/ELEMENT	DESCRIPTION
`replace-tag`	`"@" source [ ";" [ type ] [ ";" [ range ] *option ] ] "@"`
`data-definition-tag`	`"@~" source CRLF *def-line "@"`
`conditional-tag`	`"@" [ group-level ] "?" source [ ";" [ type ] [ ";" [ range ] *option ] ] "@"`
`loop-tag`	`“@" [ group-level ] "*" source [ ";" [ type ] [ ";" [ range ] *option ] ] "@"`
`loop-target-tag`	`"@.@"`
`block-tag`	`"@" [ group-level ] "[" *option "@"`
`block-termination-tag`	`"@" [ group-level ] "]@"`
`line-continuation-tag`	`"@\"`
`instruction-tag`	`"@!" *option "@"`
`single-line-comment`	`"@#" string CRLF`

Table 9: CML Grammar

CML TAG/ELEMENT	DESCRIPTION
multi-line-comment	"@##" *[string / CRLF] "#@"
def-line	type-line / range-line / option-line / printable-line / desc-line
type-line	"type" WSP "=" WSP type-elem CRLF
range-line	"range" WSP "=" WSP range CRLF
option-line	option-elem CRLF
printable-line	"printable" WSP "=" WSP string CRLF
desc-line	"description" WSP "=" *[WSP string CRLF]
group-level	int
source	absolute-path / relative-path / local-path
absolute-path	"/" path-component* name
relative-path	[path-component*] name
path-component	(name / sequence-id) "/"
sequence-id	int
local-path	"." name
name	string
type	sequence / type-elem
sequence	[order "-"] type-elem "-" sequence-elem
sequence-elem	"set" / "list"
type-elem	"int" / "string" / "ip" / "port" / "file" / etc...
order	ordered" / "unordered"
range	and-range *["," and-range]
and-range	range-elem *["&" range-elem]
range-elem	numeric-range / string range

Table 9: CML Grammar

CML TAG/ELEMENT	DESCRIPTION
numeric-range	gt-range / ge-range / lt-range / le-range / eq-range
string range	string-literal / regular-exp
gr-range	int ">"
ge-range	int ">="
lt-range	">" int
le-range	">=" int
eq-range	"=" int
string-literal	<"> string <">
regular-exp	"r" <"> string <">
option	";" option-elem
option-elem	option-name / option-nv
option-nv	option-nv
option-name	string
option-value	string