Configuring Server-side Extraction

Applies To

Software	Version
Acrolinx Server	4.5, 4.6

Server-side extraction is the process of configuring the server to extract text instead of the Acrolinx client software. Server-side extraction is useful for customized Acrolinx integrations where there's no user interface for configuring text extraction settings. For example, Acrolinx integrations for rich text editors such as CKEditor use server-side extraction.

You define server-side extraction settings by adding context segmentation definition (CSD) files to your server installation and updating your language configuration file.

To configure server-side extraction, follow these steps:

1. Create a server-side CSD file
2. Associate the server-side CSD file with a language or rule set
3. *(Optional)* Configure Acrolinx Server to generate an annotated copy of the checked document .
4. Enable server-side extraction in your client software
   Restriction: Server-side extraction should not be used when the server is configured to save Acrolinx Scorecards on the client computer. For more information about this setting, see the *Acrolinx Server Administration Guide* .

Context Segmentation Definitions (CSDs)

What is the Purpose of CSDs?

CSDs provide Acrolinx with instructions on how to get the text out of a certain kind of document. These instructions fall into two basic categories: segmentation and filtering.

When you check HTML or XML files, the settings in a CSD file determine the segmentation rules.

The CSD contains the following settings:

- The document types that are associated with the defined segmentation settings.
- Sentence-break elements.
- No-break elements.
- Elements to include or exclude when a check is performed.
- Attributes that have values which should be checked.

CSD File Naming and Location

CSDs are contained in files that have the following naming format:

`<FILENAME>.properties`.

Server-side CSD files usually are located in:

- %ACROLINX_CONFIGURATION_ROOT%\data\ on Windows.
- $ACROLINX_CONFIGURATION_ROOT/data/ on Unix-based operating systems.

Creating and Modifying CSDs

To create or modify a CSD file, follow these steps:

1. If you don't have any CSD files, create one first

   a. Create a text file `<FILENAME>.properties` and save the file in your server configuration directory.

      Example: %ACROLINX_CONFIGURATION_ROOT%\data\ `<LANGUAGE_ID>`

   b. Copy the following lines into the text file:

```
### General Information

csd_name=
version=1

###  Matching Criteria

dtd.type=
dtd.source_reference=
dtd.client_signature=
dtd.public_id=
dtd.system_id=
dtd.root_element=
dtd.schema=

###  Extraction Settings

# General Settings

ignoreCase=true
default_inclusion_mode=include

# Filtering
exclusion_elements=
inclusion_elements=
empty_elements=

# Break elements
no_break_elements=
sentence_break_elements=
token_break_elements=
default_break_level=

# Attribues
extract_attributes=
meta_information=

# Parenthetic elements
parenthetic_elements=
```

2. If you have an existing CSD that you'd like to edit, find the directory where your CSD profiles are stored and open the file:

   Example: %ACROLINX_CONFIGURATION_ROOT%\data\ `common\myCSD.properties`

3. Update the values for each of the settings.

   If you're not sure what some of the settings are for, see the descriptions of each setting.

   You can also use the following sample CSD to see how you could enter the different settings.
   Note: We've provided a sample value for each of the matching criteria, but in reality you don't need to set all these criteria. The `dtd.type` is required but the others are optional.

```
### General Information

csd_name=My Test CSD
version=1

###  Matching Criteria

dtd.type=XML
dtd.source_reference=.*/user-guides/.*.dita
dtd.language=en
dtd.client_signature=mF0YZzgQ233lY2tlcg
dtd.public_id=-//OASIS//DTD DITA Concept//EN
dtd.system_id=concept.dtd
dtd.root_element=concept
dtd.schema=notes.xsd

###  Extraction Settings

# General Settings
ignoreCase=true
default_inclusion_mode=include
mark_excluded_elements=false

# Filtering
exclusion_elements=metadata,cndname,codeph
inclusion_elements=ph,entry,title
empty_elements=img,xref

# Break elements
no_break_elements=sub,sup
sentence_break_elements=title,entry,li
token_break_elements=cr
default_break_level=sentence

# Parenthetic elements
parenthetic_elements=footnote,endnote

# Attributes
extract_attributes=img.?alt
meta_information=MyDocumentID:<book id>
```

4. Save your changes and if you haven't done so already, associate the CSD with a rule set.

Configuring the Matching Criteria and General Settings for a CSD

Matching Criteria

The properties in a CSD file fall into two main categores: the matching criteria and the extraction settings themselves.

When you create a CSD, it's usually intended for a specific type of document. For example, you might create separate CSDs for help tasks and localization files. You use the matching criteria to match the appropriate documents with your CSD. Usually, you only need to define two or three matching criteria.

Use the following table to decide which matching criteria best suits your requirements.

Table 1. Matching Criteria and General Settings

Setting	`csd_name`
Description	A name that helps to differentiate the CSD from other CSDs in case you're reviewing the log files for CSD behavior.
Values	A unique name. You can give your CSD any name you like. `csd_name=CSD for Me`

Setting	`version`
Description	This is an internal version number that indicates now new the CSD is. The server uses this information to interpret the CSD as a legacy CSD type or a new CSD type.
Values	A number. In this case, the number `1`. Just enter the number `1` as follows. `version=1` That's it, job done.

Setting	`dtd.type`
Description	Describes the type of file that was checked. The format could be text, html or xml.
Values	One of following values: `HTML`, `XML`, or `WORD_XML` To assign settings to XML files, you would enter the `dtd.type` identifier as follows: `dtd.type=XML`

Setting	`dtd.source_reference`
Description	The location of the file or files that the CSD should apply to. The server examines full path to the checked file. This path is sent by the Acrolinx plug-ins. You might use this property when there is no other data available to identify the file type.

Values	A file path or a path pattern entered as a regular expression.
	
	For example, to have the CSD apply to all DITA files in the "user-guides" directory, you would enter the property as follows:
	
	`dtd.source_reference=.*/user-guides/.*.dita`
	
	If you had multiple directories such as "user-guides" and "faqs" you would separate them with the OR operator " `|` " like this:
	
	`dtd.source_reference=.*/user-guides/.*.dita|.*/faqs/.*.dita`
	
	CAUTION:
	If you want to use more complex regular expressions, there a few things you should look out for. See our detailed section on regular expressions in matching criteria.

Setting	`dtd.language`
Description	The language of the check as indicated by the Acrolinx plug-in. You can use this property change the extraction settings based on the language the text that's being checked. One scenario might be that you're checking a multilingual XML format such as TMX. You would want to include the parts of the file that match the checking language.

Values	A shortcode for the language which is usually two letters such "en" or "de".
	
	For example, to target the German segments in a TMX file, you would enter to the property as follows:
	
	`dtd.language=de`
	
	In the same CSD, you might configure the inclusion elements property like this:
	
	`inclusion_elements=<tuv lang=de>`

Setting	`dtd.client_signature`
Description	The signature that an Acrolinx plug-in sends when it authenticates with an Acrolinx server. You might use this identifier for customized Acrolinx integrations such as CMS integrations that don?t send a plug-in short name. CAUTION: Only use this matching criterion if no other matching criteria work for you. Matching on the signature means that the CSD will only match when writers check from a certain editor. That's not really how CSDs are supposed to match but it can be useful if the host editor is the only way of determining the document type.
Values	A signature string. For example, to assign settings to an Acrolinx integration that uses the signature "xmF0YZzgQ233lY2tlcg", you would enter the `clientSignature` element as follows: `dtd.client_signature=xmF0YZzgQ233lY2tlcg`

Setting	`dtd.public_id`

Description	The public ID of an XML document type.
Values	A public ID string. For example, to assign settings to DITA concepts, you would enter the `publicId` identifier as follows: `dtd.public_id=-//OASIS//DTD DITA Concept//EN<`

Setting	`dtd.system_id`
Description	The system ID of an XML document type.
Values	A system ID string. For example, to assign settings to DITA concepts, you would enter the `systemId` identifier as follows: `dtd.system_id=concept.dtd`

Setting	`dtd.root_element`
Description	The root element of an XML or HTML document. You might use this identifier for simple XML documents that don?t have a document type definition or schema.
Values	An element name. For example, to assign settings to XML files that begin with the element `<product>`, you might enter the `rootElement` element as follows: `dtd.root_element=product` You can also use this identifier to assign settings to HTML files. The root element for HTML documents is always `html`.

Setting	`dtd.schema`
Description	The name of an XML document schema.
Values	A schema name. For example, to assign settings to XML files that use the schema "notes", you would enter the `schemaName` identifier as follows: `dtd.schema=notes.xsd`

Table 1. Matching Criteria and General Settings

Setting	Description	Values
`csd_name`	A name that helps to differentiate the CSD from other CSDs in case you're reviewing the log files for CSD behavior.	A unique name. You can give your CSD any name you like. `csd_name=CSD for Me`
`version`	This is an internal version number that indicates now new the CSD is. The server uses this information to interpret the CSD as a legacy CSD type or a new CSD type.	A number. In this case, the number `1`. Just enter the number `1` as follows. `version=1` That's it, job done.
`dtd.type`	Describes the type of file that was checked. The format could be text, html or xml.	One of following values: `HTML, XML,` or `WORD_XML` To assign settings to XML files, you would enter the `dtd.type` identifier as follows: `dtd.type=XML`

Setting	Description	Values

`dtd.source_reference`	The location of the file or files that the CSD should apply to.
	
	The server examines full path to the checked file. This path is sent by the Acrolinx plug-ins.
	
	You might use this property when there is no other data available to identify the file type.
		A file path or a path pattern entered as a regular expression.
		
		For example, to have the CSD apply to all DITA files in the "user-guides" directory, you would enter the property as follows:
		
		`dtd.source_reference=.*/user-guides/.*.dita`
		
		If you had multiple directories such as "user-guides" and "faqs" you would separate them with the OR operator " `|` " like this:
		
		`dtd.source_reference=.*/user-guides/.*.dita|.*/faqs/.*.dita`
		
		CAUTION:
		If you want to use more complex regular expressions, there a few things you should look out for. See our detailed section on regular expressions in matching criteria.

`dtd.language`	The language of the check as indicated by the Acrolinx plug-in.
	
	You can use this property change the extraction settings based on the language the text that's being checked.
	
	One scenario might be that you're checking a multilingual XML format such as TMX. You would want to include the parts of the file that match the checking language.
		A shortcode for the language which is usually two letters such "en" or "de".
		
		For example, to target the German segments in a TMX file, you would enter to the property as follows:
		
		`dtd.language=de`
		
		In the same CSD, you might configure the inclusion elements property like this:
		
		`inclusion_elements=<tuv lang=de>`

`dtd.client_signature`	The signature that an Acrolinx plug-in sends when it authenticates with an Acrolinx server. You might use this identifier for customized Acrolinx integrations such as CMS integrations that don?t send a plug-in short name. CAUTION: Only use this matching criterion if no other matching criteria work for you. Matching on the signature means that the CSD will only match when writers check from a certain editor. That's not really how CSDs are supposed to match but it can be useful if the host editor is the only way of determining the document type.	A signature string. For example, to assign settings to an Acrolinx integration that uses the signature "xmF0YZzgQ233lY2tlcg", you would enter the `clientSignature` element as follows: `dtd.client_signature=xmF0YZzgQ233lY2tlcg`
`dtd.public_id`	The public ID of an XML document type.	A public ID string. For example, to assign settings to DITA concepts, you would enter the `publicId` identifier as follows: `dtd.public_id=-//OASIS//DTD DITA Concept//EN<`
`dtd.system_id`	The system ID of an XML document type.	A system ID string. For example, to assign settings to DITA concepts, you would enter the `systemId` identifier as follows: `dtd.system_id=concept.dtd`
`dtd.root_element`	The root element of an XML or HTML document. You might use this identifier for simple XML documents that don?t have a document type definition or schema.	An element name. For example, to assign settings to XML files that begin with the element `<product>`, you might enter the `rootElement` element as follows: `dtd.root_element=product` You can also use this identifier to assign settings to HTML files. The root element for HTML documents is always `html`.
`dtd.schema`	The name of an XML document schema.	A schema name. For example, to assign settings to XML files that use the schema "notes", you would enter the `schemaName` identifier as follows: `dtd.schema=notes.xsd`

Using Regular Expressions in Matching Criteria

You can enter a regular expression as the value for any matching criteria. Regular exexpressions are especially useful for the criterion `dtd.source_reference` because you can use them to define multiple file locations.

When you use regular expressions, bear in mind that you have to use the syntax for Java-style regular expressions.

If you enter Windows-style paths, you must also escape the backslashes. In the Java-style regular expressions , you must escape backslashes twice. For example, suppose that your files are stored in the master directory `C:\DOC\user-guides\`. To match all DITA files in the directory you would enter the regular expression as follows:

```
dtd.source_reference=C:\\\\DOC\\\\user-guides\\\\.*.dita
```

Because of the idiosyncrasies of Java-style regular expressions, you have to replace each single backslash "\" with four backslahes "\\\\".

Configuring Extraction Settings in a CSD

After you're familiar with the different ways you can categorize elements in Acrolinx, you can start adding these elements to your extraction settings.

If you want to change the text extraction for elements that have specific attributes, you can add the attribute to your settings as well. For example, you could categorize `p` elements as excluded only when they have the value `"internal"` in the `audience` attribute. You add attributes to your CSD in a specific syntax.

Use the following table as a guide to what you can configure in a CSD and how you add the extraction settings correctly:

Table 2. Extraction Settings

Setting	`ignoreCase`
Description	If your element names appear in different cases you might want to turn this off. By default, Acrolinx ignores the casing of element names when assessing the extraction settings. For example, if you configure the element name `title` to be excluded, Acrolinx also matches the elements `TITLE` and `Title`.
Values	Boolean value. *Example*: `true, false` Suppose that you have an element such as `Data` and it means something different when it takes the uppercase form `DATA`. In this case, you would configure the extractions settings to be case-sensitive by entering the property as follows: `ignoreCase=false`
Setting	`exclusion_elements`
Description	If you don't want Acrolinx to check the text in an element at all, define it as excluded.

Values	Element names, element names with attributes.
	Example: `metadata, cndname,codeph`
	For example, you probably don't want to check the text in elements that contain code or identifiers.
	`<metadata>document ID12345</metadata>`
	Note: Excluded XML elements that are nested within included elements are still excluded. All other excluded items are ignored.
	In the following example, the `<note>` element is included and the `<ph>` element is excluded.
	`<note>Thistextwillbechecked.<ph>Thistextwillnotbechecked.</ph></note>`
	The content in `<ph>` will still be excluded, even if it?s contained within an included element.

Setting	`inclusion_elements`
Description	Use the property to define elements that are always included regardless of whether they're nested inside an excluded element. "Child" elements of inclusion elements are also always included.
Values	Element names, element names with attributes. For example, to always include `title` elements in a check, add it to your list of inclusion elements like this: `inclusion_elements=ph, entry, title`
Setting	`sentence_break_elements`
Description	If you have elements that contain sentences that don't end with a period, define the as sentence-break elements. This means that the end of the element should always be treated as a sentence end.

Values	Element names, element names with attributes.
	A very typical example is the title element:
	`<title>This is a title</title>`
	Unexpected `sentence_too_long` flags which actually consist of several sentences are a clear sign that one or more elements must be defined as a sentence break.
	In this case, you would add `title` to the list of sentence-break elements like this:
	`sentence_break_elements=title,entry,li`

Setting	`no_break_elements`
Description	If you have elements that split a word into separate fragments, define it as a "no break" element.

Values	Element names, element names with attributes.
	A typical example is the subscript element:
	`H<sub>2</sub>O concentration`
	If `<sub>` isn?t classified, the incorrect text
	`H 2 O concentration`
	is sent to the server. If `<sub>` is a no break element, the correct text
	`H2O concentration`
	is sent to the server.
	In this case, you would add `sub` to the list of no break elements like this:
	`no_break_elements=sub,sup`

Setting	`token_break_elements`
Description	You might define elements as token breaks of you have words which aren't separated by a space. This setting defines the elements which should cause a token break. Adding a token break is basically adding a boundary between words.

Values	Element names, element names with attributes.
	Usually, you would add a space between words, but sometimes editors separate words with elements rather than spaces.
	For example, WordProcessingML uses `<cr/>`elements to indicate manually inserted line breaks, like this:
	`This sentence is<cr/>wrapped manually.`
	You wouldn't want the sentence to end at "is", or have the words interpreted as "iswrapped". So you would update the setting as follows:
	`token_break_elements=cr`

Setting	`default_break_level`
Description	This setting defines how Acrolinx should interpret elements that aren't already included in the "break element" settings or in any of the other settings. The following settings are "break element" settings: • `no_break_elements` • `sentence_break_elements` • `token_break_elements`
Values	Enter one of the following values: • `none` Acrolinx doesn't add any break at all. • `token` Acrolinx adds a token break. • `sentence` Acrolinx adds a sentence break. The default value for this setting is `token`. For example, if you wanted to have any undefined elements treated as sentence breaks, you would enter the property as follows: `default_break_level=sentence`

Setting	`parenthetic_elements`
Description	Suppose that you have elements that surround fragments of text in the same way that you would put a side-note in parentheses in the middle of a sentence. Example: `<p>This is the <footnote>This is a second sentence.</footnote> first sentence.</p>` This type of element is fairly rare and should be classified as a **parenthetical element** in the segmentation options.

Values	Element names.
	A typical example is a footnote in the middle of a sentence.
	If `<footnote>` isn?t classified as a **parenthetical element**, the incorrect text
	`This is the This is a secondsentence.first sentence.`
	is sent to the server. If `<footnote>` is classified as a parenthetical element, the correct text
	`This is the first sentence. This is a second sentence.`
	is sent to the server.
	In this case, you would add `footnote` to the list of parenthetical elements like this:
	`parenthetic_elements=footnote,endnote`

Setting	`empty_elements`

Description	If you have elements that don't contain any text such as text entities, but should still be treated as part of the sentence, define them as empty elements.
	There are several things to consider when working with elements that don't contain any text:
	• The plug-in replaces these elements with a placeholder character when processing the text for checking.
	Sometimes elements that don't contain any text but are part of the sentence structure can cause style or grammar rules to flag incorrectly.
	In the following example, the `xref` element contains no text but acts as a noun in the sentence.
	`<para>Slidethe<xref xidtype="IBSwitch" xrefid="ib001"></xref>firmlyintoplace.</para>`
	By default, the text "Slide the firmly into place." is sent to the server and the sentence is flagged for incorrect grammar. When you classify the elements as empty, you prevent this type of issue because empty elements are treated as nouns.
	• Acrolinx always ignores the contents of empty elements.
	This fact is important to consider if your empty element is usually empty but occasionally does contain text.
	Considering the previous example, suppose that your xrefs are usually empty but you do have one xref that spans a fragment of your sentence.
	`<para>Slidethe<xref xidtype="IBSwitch" xrefid="ib001">first switch</xref>firmlyintoplace.</para>`
	The text that is within the `xref` tag will be ignored.
	If you happened to misspell the word "first" as "frst", Acrolinx would not find this spelling issue because the text is contained within an empty element.

Values	Element names.
	For example, images are sometimes used in place of words like in the following excerpt:
	`Powered by <image href="acrolinx_logo.jpg"></image>.`
	In this case, you would add `img` to the list of empty elements like this:
	`empty_elements=img,xref`

Setting	`default_inclusion_mode`
Description	When you run a check, this setting determines whether elements are included or excluded by default. The following elements are *always* excluded or included, regardless of the `default_inclusion_mode` value: • Elements that are defined as `exclusion_elements` or `inclusion_elements` • "Child" elements of exclusion or inclusion elements.

Values	One of the following mode names: • `include` • `exclude` For example, if you would prefer to have all elements excluded by default unless you explicitly include them, you would enter the property as follows: `default_inclusion_mode=exclude`
Setting	`mark_excluded_elements`
Description	Configure the server to insert placeholders for excluded elements when processing the text. The default is false. This property prevents incorrect flags that are caused by excluded elements. For more information on this what this property does, see the Batch Checker documentation.
Values	Boolean value. *Example*: `true, false`
Setting	`extract_attributes`
Description	Use this property to configure Acrolinx to include the values of attributes when sending text to the server for checking. You can use this feature in combination with customized style rules to validate the values of specific attributes. You can also use this feature to check attributes that contain sentences or keywords such as meta content. For more information on developing style rules to validate attribute values, contact your Acrolinx project consultant
Values	Attribute names, element names with attribute names. You can define how attributes are extracted in several ways: • Extract the content of a specific attribute regardless of the element in which it appears. Syntax: `.<AttributeName>` The following example shows how to configure the Batch Checker to extract and check the value of the `name` attribute: `extract_attributes=.name` This configuration would extract the text "Header" and "Checking Statistics" from the following code excerpt. `<img name="Header" src="header.jpg /> <tablename="Checking statistics">` • Extract the content of a specific attribute when it?s found in a specific element Syntax: `<ElementName>.<AttributeName>` The following example shows how to configure the Batch Checker to extract and check the value of the `alt` attribute within image elements: `extract_attributes=img.alt` This configuration would extract the text "Options Menu" and "Configuration Window" from the following code excerpt. `<imgalt="Options Menu" src="13682.jpg"/> <imgalt="Configuration Window" src="3738.jpg"/>` It?s also possible to develop style rules to check for empty attributes so that you can check for images where the `alt` attribute exists but isn?t defined. • Extract the content of a specific attribute only when the element contains another attribute with a specific value. This syntax is useful for checking the content of meta tags `<ElementName>.<AttributeName>#<restrictingAttribute>=<restrictingValue>` The following example shows how to configure the Batch Checker to extract and check the value of the `content` attribute from all `meta` elements as long as the `meta` element has the name "description". *Example:* `extract_attributes=meta.content#name=description` This configuration would extract the text "Topspin is about making your business better." from the following code excerpt. `<meta name="Description" content="Topspin is about making your business better." />`
Setting	`meta_information`

Description	Some Acrolinx clients such as the **Acrolinx Batch Checker** can read the contents of certain elements or attributes within checked XML documents and write those values into the Scorecards as metadata. This functionality can be useful to process the Scorecards offline and to aggregate the report data by specific custom values. For more information about this feature, see the Batch Checker documentation.
Values	The input for this parameter is set with two values • The value **Meta Attribute Name** defines how a meta tag appears in the Scorecard. The Batch Checker writes the **Meta Attribute Names** as attributes into the `<identifier>` tag (prefixed with "meta_"). The **Meta Attribute Name** enables you to label your metadata. • The value **Source Element** tells the Batch Checker which elements to look for in a checked XML document, and to insert the element values into new meta attributes of the `<identifier>` tag. Enter the **Meta Attribute Name** value first, followed by a colon and then enter the **Source Element Name** surrounded by angled brackets: `<...>`. *Example:* `meta_information=MetaTagName:<sourceElement>` The **Source Element** value can also reference **Attributes** of XML elements. To define an **Attribute** of a **Source Element**, enter the attribute name after the source element name separated by a space: *Example:* `meta_information=MetaTagName:<sourceElementsourceAttribute>` The `meta_information` tag can also take multiple value pairs separated by a semicolon: *Example:* `meta_information=MyDocumentID:<bookid>;MyRoleName:<bookelementrole>`

Table 2. Extraction Settings

Setting	Description	Values
`ignoreCase`	If your element names appear in different cases you might want to turn this off. By default, Acrolinx ignores the casing of element names when assessing the extraction settings. For example, if you configure the element name `title` to be excluded, Acrolinx also matches the elements `TITLE` and `Title`.	Boolean value. *Example*: `true, false` Suppose that you have an element such as `Data` and it means something different when it takes the uppercase form `DATA`. In this case, you would configure the extractions settings to be case-sensitive by entering the property as follows: `ignoreCase=false`

`exclusion_elements`	If you don't want Acrolinx to check the text in an element at all, define it as excluded.	Element names, element names with attributes.
		
		Example: `metadata, cndname,codeph`
		
		For example, you probably don't want to check the text in elements that contain code or identifiers.
		
		`<metadata>document ID12345</metadata>`
		
		Note: Excluded XML elements that are nested within included elements are still excluded. All other excluded items are ignored.
		
		In the following example, the `<note>` element is included and the `<ph>` element is excluded.
		
		`<note>Thistextwillbechecked.`
		`<ph>Thistextwillnotbechecked.</ph></note>`
		
		The content in `<ph>` will still be excluded, even if it?s contained within an included element.

`inclusion_elements`	Use the property to define elements that are always included regardless of whether they're nested inside an excluded element. "Child" elements of inclusion elements are also always included.	Element names, element names with attributes. For example, to always include `title` elements in a check, add it to your list of inclusion elements like this: `inclusion_elements=ph, entry, title`

`sentence_break_elements`	If you have elements that contain sentences that don't end with a period, define the as sentence-break elements. This means that the end of the element should always be treated as a sentence end.	Element names, element names with attributes.
		A very typical example is the title element:
		
		`<title>This is a title</title>`
		
		Unexpected `sentence_too_long` flags which actually consist of several sentences are a clear sign that one or more elements must be defined as a sentence break.
		
		In this case, you would add `title` to the list of sentence-break elements like this:
		
		`sentence_break_elements=title,entry,li`

Setting	Description	Values

`no_break_elements`	If you have elements that split a word into separate fragments, define it as a "no break" element.	Element names, element names with attributes.
		A typical example is the subscript element:
		`H<sub>2</sub>O concentration`
		If `<sub>` isn?t classified, the incorrect text
		`H 2 O concentration`
		is sent to the server. If `<sub>` is a no break element, the correct text
		`H2O concentration`
		is sent to the server.
		In this case, you would add `sub` to the list of no break elements like this:
		`no_break_elements=sub,sup`

`token_break_elements`	You might define elements as token breaks of you have words which aren't separated by a space.
	This setting defines the elements which should cause a token break. Adding a token break is basically adding a boundary between words.
		Element names, element names with attributes.
		Usually, you would add a space between words, but sometimes editors separate words with elements rather than spaces.
		For example, WordProcessingML uses `<cr/>`elements to indicate manually inserted line breaks, like this:
		`This sentence is<cr/>wrapped manually.`
		You wouldn't want the sentence to end at "is", or have the words interpreted as "iswrapped". So you would update the setting as follows:
		`token_break_elements=cr`

`default_break_level`	This setting defines how Acrolinx should interpret elements that aren't already included in the "break element" settings or in any of the other settings. The following settings are "break element" settings: • `no_break_elements` • `sentence_break_elements` • `token_break_elements`	Enter one of the following values: • `none` Acrolinx doesn't add any break at all. • `token` Acrolinx adds a token break. • `sentence` Acrolinx adds a sentence break. The default value for this setting is `token`. For example, if you wanted to have any undefined elements treated as sentence breaks, you would enter the property as follows: `default_break_level=sentence`

`parenthetic_elements`	Suppose that you have elements that surround fragments of text in the same way that you would put a side-note in parentheses in the middle of a sentence.
	Example:
	`<p>This is the <footnote>This is a second sentence.</footnote> first sentence.</p>`
	This type of element is fairly rare and should be classified as a **parenthetical element** in the segmentation options.
		Element names.
		A typical example is a footnote in the middle of a sentence.
		If `<footnote>` isn?t classified as a **parenthetical element**, the incorrect text
		`This is the This is a secondsentence.first sentence.`
		is sent to the server. If `<footnote>` is classified as a parenthetical element, the correct text
		`This is the first sentence. This is a second sentence.`
		is sent to the server.
		In this case, you would add `footnote` to the list of parenthetical elements like this:
		`parenthetic_elements=footnote,endnote`

`empty_elements`	If you have elements that don't contain any text such as text entities, but should still be treated as part of the sentence, define them as empty elements. There are several things to consider when working with elements that don't contain any text:	Element names. For example, images are sometimes used in place of words like in the following excerpt: `Powered by <image href="acrolinx_logo.jpg"></image>.` In this case, you would add `img` to the list of empty elements like this:

Setting	Description	Values

	• The plug-in replaces these elements with a placeholder character when processing the text for checking.
	
	Sometimes elements that don't contain any text but are part of the sentence structure can cause style or grammar rules to flag incorrectly.
	
	In the following example, the `xref` element contains no text but acts as a noun in the sentence.
	
	`<para>Slidethe<xref xidtype="IBSwitch" xrefid="ib001"></xref>firmlyintoplace.</para>`
	
	By default, the text "Slide the firmly into place." is sent to the server and the sentence is flagged for incorrect grammar. When you classify the elements as empty, you prevent this type of issue because empty elements are treated as nouns.
	
	• Acrolinx always ignores the contents of empty elements.
	
	This fact is important to consider if your empty element is usually empty but occasionally does contain text.
	
	Considering the previous example, suppose that your xrefs are usually empty but you do have one xref that spans a fragment of your sentence.
	
	`<para>Slidethe<xref xidtype="IBSwitch" xrefid="ib001">first switch</xref>firmlyintoplace.</para>`
	
	The text that is within the `xref` tag will be ignored.
	
	If you happened to misspell the word "first" as "frst", Acrolinx would not find this spelling issue because the text is contained within an empty element.
		`empty_elements=img,xref`

Setting	Description	Values
`default_inclusion_mode`	When you run a check, this setting determines whether elements are included or excluded by default. The following elements are *always* excluded or included, regardless of the `default_inclusion_mode` value: • Elements that are defined as `exclusion_elements` or `inclusion_elements` • "Child" elements of exclusion or inclusion elements.	One of the following mode names: • `include` • `exclude` For example, if you would prefer to have all elements excluded by default unless you explicitly include them, you would enter the property as follows: `default_inclusion_mode=exclude`
`mark_excluded_elements`	Configure the server to insert placeholders for excluded elements when processing the text. The default is false. This property prevents incorrect flags that are caused by excluded elements. For more information on this what this property does, see the Batch Checker documentation.	Boolean value. *Example*: `true, false`

`extract_attributes`	Use this property to configure Acrolinx to include the values of attributes when sending text to the server for checking.
	
	You can use this feature in combination with customized style rules to validate the values of specific attributes. You can also use this feature to check attributes that contain sentences or keywords such as meta content. For more information on developing style rules to validate attribute values, contact your Acrolinx project consultant
		Attribute names, element names with attribute names.
		You can define how attributes are extracted in several ways:
		
		• Extract the content of a specific attribute regardless of the element in which it appears.
		
		Syntax: `.<AttributeName>`
		
		The following example shows how to configure the Batch Checker to extract and check the value of the `name` attribute:
		
		`extract_attributes=.name`
		
		This configuration would extract the text "Header" and "Checking Statistics" from the following code excerpt.
		
		`<img name="Header" src="header.jpg />`
		
		`<tablename="Checking statistics">`
		
		• Extract the content of a specific attribute when it?s found in a specific element
		
		Syntax: `<ElementName>.<AttributeName>`
		
		The following example shows how to configure the Batch Checker to extract and check the value of the `alt` attribute within image elements:
		
		`extract_attributes=img.alt`
		
		This configuration would extract the text "Options Menu" and "Configuration Window" from the following code excerpt.
		
		`<imgalt="Options Menu" src="13682.jpg"/>`
		
		`<imgalt="Configuration Window" src="3738.jpg"/>`
		
		It?s also possible to develop style rules to check for empty attributes so that you can check for images where the `alt` attribute exists but isn?t defined.

Setting	Description	Values

		• Extract the content of a specific attribute only when the element contains another attribute with a specific value. This syntax is useful for checking the content of meta tags
		
		`<ElementName>.<AttributeName>#<restrictingAttribute>=<restrictingValue>`
		
		The following example shows how to configure the Batch Checker to extract and check the value of the `content` attribute from all `meta` elements as long as the `meta` element has the name "description".
		
		Example:
		`extract_attributes=meta.content#name=description`
		
		This configuration would extract the text "Topspin is about making your business better." from the following code excerpt.
		
		`<meta name="Description" content="Topspin is about making your business better." />`

`meta_information`	Some Acrolinx clients such as the **Acrolinx Batch Checker** can read the contents of certain elements or attributes within checked XML documents and write those values into the Scorecards as metadata. This functionality can be useful to process the Scorecards offline and to aggregate the report data by specific custom values. For more information about this feature, see the Batch Checker documentation.	The input for this parameter is set with two values • The value **Meta Attribute Name** defines how a meta tag appears in the Scorecard. The Batch Checker writes the **Meta Attribute Names** as attributes into the `<identifier>` tag (prefixed with "meta_"). The **Meta Attribute Name** enables you to label your metadata. • The value **Source Element** tells the Batch Checker which elements to look for in a checked XML document, and to insert the element values into new meta attributes of the `<identifier>` tag. Enter the **Meta Attribute Name** value first, followed by a colon and then enter the **Source Element Name** surrounded by angled brackets: `<...>`. *Example:* `meta_information=MetaTagName:<sourceElement>` The **Source Element** value can also reference **Attributes** of XML elements. To define an **Attribute** of a **Source Element**, enter the attribute name after the source element name separated by a space: *Example:* `meta_information=MetaTagName:<sourceElementsourceAttribute>` The `meta_information` tag can also take multiple value pairs separated by a semicolon: *Example:* `meta_information=MyDocumentID:<bookid>;MyRoleName:<bookelementrole>`

Syntax for Entering Attributes in a CSD

Most settings in a CSD file take the element names as their values, like this: `sentence_break_elements=codeph,cmdname`. However, you can also enter elements in combination with attributes. If you want to select elements based on their attributes, you enter the values in a specific syntax.

Syntax for Entering Attributes

Acrolinx can select elements for segmentation based on their attributes. You can then refine the segmentation behavior to respond to specific conditions. To enter an attribute as a value of a segmentation parameter, use the following syntax:

- `<elementNameattributeName>` - This value instructs Acrolinx to segment the element if it contains the specified attribute.
- `<elementNameattributeName="value">` - This value instructs Acrolinx to segment the element only if it contains the specified attribute with the specified value.
  Remember: This syntax is case-sensitive. Ensure that the case of the elements, attributes, and attribute values match the case used in your XML documents. This syntax is used to segment text within elements only. To include the values of attributes for checking, you use a different configuration.

The following example shows an exclusion element parameter set to exclude two elements based on their attributes.

```
exclusion_elements=<MODULEmodname="voicecontrol">,<PROCEDUREflag>
```

The first value tells Acrolinx to exclude text that is located in `module` elements that have a `modname` attribute set to `voicecontrol`. The second value tells Acrolinx to exclude text within `procedure` elements whenever the element contains the `flag` attribute, regardless of the attribute value.

To include the values of attributes for checking, you use a different syntax.