Platform SDK Developer's Guide

Lazy Parsing of Message Attributes

4/10/2025

Contents

Lazy Parsing of Message Attributes

This page provides:

- an overview and list of requirements for the lazy parsing feature
- design details explaining how this feature works
- code examples showing how to implement lazy parsing in your applications

Introduction to Lazy Parsing

Lazy parsing allows users to specify which attributes should always be parsed immediately, and which attributes should be parsed only on demand.

Some complex attributes (such as the `ConfObject` attribute found in some Configuration Server protocol messages) are large and very complex. Unpacking these attributes can be time-consuming and, in cases when an application is not interested in that data, can affect program performance. This issue is resolved by using the "lazy parsing" feature included with the Platform SDK 8.1 release, which is described in this article.

When this feature is turned off, all message attributes are parsed immediately - which is normal behavior for previous version of the Platform SDK. When lazy parsing is enabled, any attributes that were tagged for lazy parsing are only parsed on demand. In this case, if the application does not explicitly check the value of an attribute tagged for lazy parsing then that attribute is never parsed at all.

Feature Overview

- Platform SDK includes configuration options to turn the lazy parsing functionality on or off for each individual protocol that supports this feature.
- Potentially time-consuming attributes that are candidates for lazy parsing are selected and marked by Platform SDK developers. Refer to your Platform SDK API Reference for details.
- To maintain backwards compatibility, there is no change in how user applications access attribute values.
- Default values:
  - In Platform SDK for Java, the lazy parsing feature is turned **on** by default.
    Note: The default value changed in release 8.1.4; in earlier releases of Platform SDK for Java, lazy parsing is turned off by default.
  - In Platform SDK for .NET, the lazy parsing feature is turned **off** by default.

Java

System Requirements

Platform SDK for Java:

- Configuration SDK protocol release 8.1 or later<ref name="ConfObjectJava">**Note:** Currently, lazy parsing is only used with the `EventObjectsRead.ConfObject` property of the Configuration Platform SDK.</ref>
- Java SE 5, 6, 7

<references/>

Design Details

This section describes the main classes and interfaces you will need to be familiar with to implement lazy parsing in your own application.

Enabling and Disabling the Lazy Parsing Feature

At any time, a running application can enable or disable lazy parsing for a specific protocol in just a few lines of code. This is done in three easy steps:

1. Create a new `KeyValueCollection` object.
2. Set the appropriate value for the `Connection.LAZY_PARSING_ENABLED_KEY` field. A value of `True` enables the feature, while `False` disables lazy parsing.
3. Use a `KeyValueConfiguration` object to apply that setting to the desired protocol(s).

> **Tip**
> Starting with release 8.1.4, the default value of the `Connection.LAZY_PARSING_ENABLED_KEY` field is always `True`, with the lazy parsing feature enabled.

Once lazy parsing mode is enabled for a protocol, the change is applied immediately. Every new message that is received takes the lazy parsing setting into account: parsing entire messages if the feature is disabled, or leaving some attributes unparsed until their values are requested if the feature is enabled.

To enable lazy parsing for the Configuration Server protocol, an application would use the following code:

```
[Java]

KeyValueCollection kv = new KeyValueCollection();
kv.addString(Connection.LAZY_PARSING_ENABLED_KEY, "true");
KeyValueConfiguration kvcfg = new KeyValueConfiguration(kv);
ConfServerProtocol cfgChannel = new ConfServerProtocol(endpoint);
cfgChannel.configure(kvcfg); //lazy parsing is immediately active after this line
```

To disable lazy parsing for the protocol only the second line of code is changed before re-applying the configuration, as shown below:

```
[Java]

kv.addString(Connection.LAZY_PARSING_ENABLED_KEY, "false");
```

.NET

System Requirements

- Configuration SDK protocol release 8.1 or later<ref name="ConfObjectNet">**Note:** Currently, lazy parsing is only used with the `EventObjectsRead.ConfObject` property of the Configuration Platform SDK.</ref>
- .NET Framework 3.5, 4.0, 4.5
- Visual Studio 2008 (required for .NET project files)

<references/>

Design Details

This section describes the main classes and interfaces you will need to be familiar with to implement lazy parsing in your own application.

Enabling and Disabling the Lazy Parsing Feature

At any time, a running application can enable or disable lazy parsing for a specific protocol in just a few lines of code. This is done in three easy steps:

1. Create a new `KeyValueCollection` object.
2. Set the appropriate value for the `CommonConnection.LazyParsingEnabledKey` field. A value of `True` enables the feature, while `False` disables lazy parsing.
3. Use a `KeyValueConfiguration` object to apply that setting to the desired protocol(s).

> **Tip**
>
> The default value of the `CommonConnection.LazyParsingEnabledKey` field is always `False`, with the lazy parsing feature disabled.

Once lazy parsing mode is enabled for a protocol, the change is applied immediately. Every new message that is received takes the lazy parsing setting into account: parsing entire messages if the feature is disabled, or leaving some attributes unparsed until their values are requested if the feature is enabled.

To enable lazy parsing for the Configuration Server protocol, an application would use the following code:

```
[C#]

KeyValueCollection kvc = new KeyValueCollection();
kvc[CommonConnection.LazyParsingEnabledKey] = "true";
KeyValueConfiguration kvcfg = new KeyValueConfiguration(kvc);
ConfServerProtocol cfgChannel = new ConfServerProtocol(endpoint);
cfgChannel.Configure(kvcfg); //lazy parsing is immediately active after this line
```

To disable lazy parsing for the protocol only the second line of code is changed before re-applying the configuration, as shown below:

```
[C#]

kvc[CommonConnection.LazyParsingEnabledKey] = "false";
```

Accessing Attribute Values

There is no difference in how applications access attribute values from returned messages. Whether the lazy parsing feature is enabled or disabled, whether the attribute being access supports lazy parsing or not, your code remains exactly the same.

However, you should consider differences in timing when accessing attribute values.

- When lazy parsing is disabled, the entire message is parsed immediately when it is received. Accessing attribute values is very fast, as the requested information is already prepared.
- When lazy parsing is enabled, the delay to parse the message upon arrival is smaller but accessing any attributes that support lazy parsing causes a slightly delay as that information must first be parsed. Note that accessing the same attribute a second time will not result in the attribute information being parsed a second time; Platform SDK saves parsed data.

Additional Notes

- XML Serialization — The `XmlMessageSerializer` class has been updated to support lazy parsing. If a message that contains unparsed attributes is serialized, then `XmlMessageSerializer` will trigger parsing before the serialization process begins.
- `ToString` method — Use of the `ToString` method does not trigger parsing of attributes that support lazy parsing. In this case, each unparsed attribute has its name printed along with a value of: "<value

is not yet parsed>".