Apply validation

Structured documents

Using Apstrata, you can define a document type that can be used to create your documents (documents are NoSQL key/value pairs that hold your content). This type, written in XML, is called a "schema". Schemas allow you to define rules that are automatically applied to documents referring to them, guaranteeing thus the consistency of the data they contain.

To create a Schema, you can invoke the SaveSchema API or, more simply, log in to the Apstrata workbench and click "Manage App > Schemas > New".

Back to the map **Next station**: control access

Example 1: the game document schema

So you are implementing this cool game that runs on mobile devices. At some point, your users need to save their game so they can come back later and start from where they left. Let us assume that what you have to save is the player's name, the current level and score, as well as the remaining "lives" of the player. You also want to make sure that:

- The value of the score is a numeric
- The level is also a numeric
- The lives field is a numeric
- The player's name is a string

So let's go ahead and write a schema. We first start with the definition of the fields. Open the workbench as recommend in the preceding paragraph, then click "Manage App > Schemas > New". The workbench opens an editor with a predefined schema template. Head to the "<fields>" section (we will get back later to the "<aclGroups>" section).

First step is to define our fields, "score", "fields" and "lives" of type "numeric", and "player" of type "string".

schema excerpt

```
...
<fields>
        <field name="score" type="numeric"/>
        <field name="level" type="numeric"/>
        <field name="lives" type="numeric"/>
        <field name="player" type="string"/>
</fields>
...
```

That's cool. Now we know that documents based on this schema will contain the aforementioned fields and that Apstrata will automatically check that the type of the content is compatible with the fields definitions.

Let us move back to the <aclGroups> section. We will not dive into it for now, just note that it is used to restrict the read/write access to your documents. For now, we will restrict read and write access on our document to the creator of that document (creator is a predefined role in Apstrata, automatically set to the user). This is how our final schema looks like:

complete schema

```
<schema>
        <aclGroups>
                <aclGroup name="the_creator">
                        <read>creator</read> <!-- creator is a predefined role in Apstrata, automatically
set to the user who creates the document -->
                        <write>creator</write>
                        <fields> <!-- the below fields are only accessible in read/write to the creator of
the document -->
                                <field>score</field>
                                <field>level</field>
                                <field>lives</field>
                                <field>player</field>
                        </fields>
                </aclGroup>
                <schemaAcl> <!-- nobody, except the application owner is entitled to read or modify the
current schema -->
                        <read>nobody</read>
                        <write>nobody</write>
                        <delete>nobody</delete>
                </schemaAcl>
        </aclGroups>
        <fields> <!-- Documents based on the current schema can contain the following fields -->
                <field name="score" type="numeric"/>
                <field name="level" type="numeric"/>
                <field name="lives" type="numeric"/>
                <field name="player" type="string"/>
</fields>
</schema>
```

Example 2: score, level, lives and player are mandatory

Since it does not make sense to save a game without a score, a level or a player name for example, how can we enforce the setting of values to these fields? Well, this is simple: we just need to specify the minimum cardinality of the corresponding field, as described in the below example.

schema excerpt

```
...
<fields>
        <field name="score" type="numeric">
                <validation>
                                <cardinality min="1" max="1"/>
                </validation>
        </field>
        <field name="level" type="numeric">
                <validation>
                                <cardinality min="1" max="1"/>
                </validation>
        </field>
        <field name="lives" type="numeric">
                <validation>
                                <cardinality min="1" max="1"/>
                </validation>
        </field>
        <field name="player" type="string">
                <validation>
                                <cardinality min="1" max="1"/>
                </validation>
        </field>
</fields>
...
```

Now that's better! Using the <validation> and <cardinality> elements, we specified for each field the minimum and maximum occurrences (note that when max is greater than one, then multiple value occurrences are allowed for the given field, which is automatically turned into an array).

Example 3: negative score is not allowed

What if you wanted automatic validation of the values that are saved in your documents? For example, you would like to have the following rules automatically verified:

- The value of the score is a numeric such as score >= 0
- The level is also a numeric such as 1 <= level <= 30 (there are 30 levels in your game)
- The lives field is a numeric such as 0 <= lives <= 10
- The player's name only contains alphanumeric characters

Here as well, we simply resort to the <validation> element:

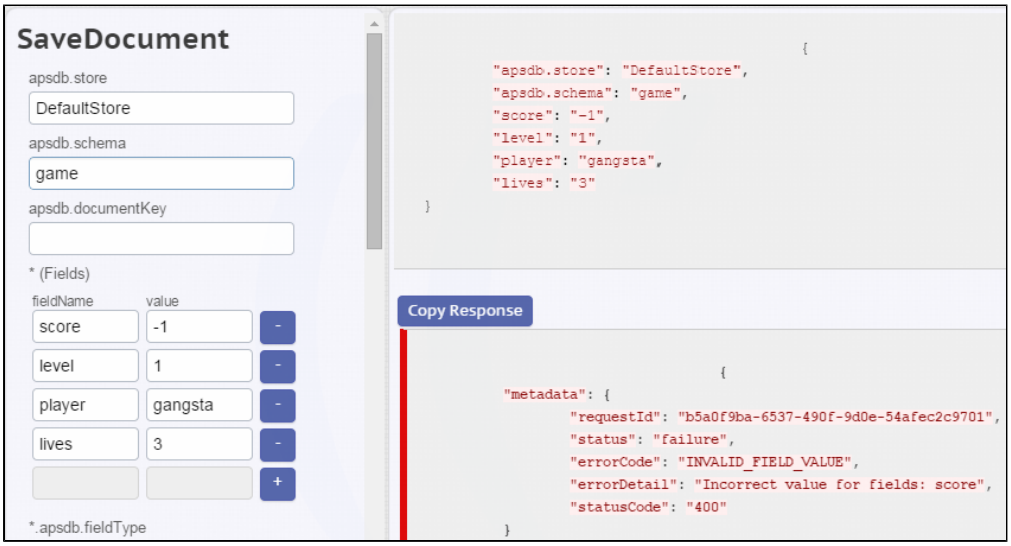
schema excerpt

```
...
<fields>
        <field name="score" type="numeric">
                <validation>
                                <cardinality min="1" max="1"/>
                                <range min="0"/> <!-- The score cannot be less than 0 -->
                </validation>
        </field>
        <field name="level" type="numeric">
                <validation>
                                <cardinality min="1" max="1"/>
                                <range min="0" max="30" /> <!-- The level is a value between 0 and 30 -->
                </validation>
        </field>
        <field name="lives" type="numeric">
                <validation>
                                <cardinality min="1" max="1"/>
                                <range min="0" max="10" /> <!-- A player can have 0 from 10 lives -->
                </validation>
        </field>
        <field name="player" type="string">
                <validation>
                                <cardinality min="1" max="1"/>
                                <regex>^([a-zA-Z0-9@*#]{3,10})$</regex> <!-- The player's name is an
alphanumeric string of 3 to 10 characters -->
                </validation>
        </field>
</fields>
...
```

Try it!

You can directly try your schema from the Apstrata workbench, using the API Explorer. Log in to the workbench, click on API Explorer then select SaveDocument.

In the form, fill in the store, apsdb.schema fields and add the score, level, lives and player parameters using the (Fields) section, by giving an invalid value to the score for example. Click on "Run" at the bottom and observe how Apstrata automatically applied the validation rules defined in your schema.



Dig deeper

- Documents
- Schema API

Related tutorials

Apply validation	Create Query components	Persist your content	Query your content

Apply validation