Translations

General Information

There are two entities that require localization:

1. Market information including markets, outcomes, periods.

Markets localization

Response is in json format with this structure:

```
{
  "markets": {
    "1": [
      {
        "condition": "=H::=4::",
        "translation": "Winner"
      },
      {
        "condition": "=H::::",
        "translation": "Match winner"
      },
    ]
  },
  "outcomes": {
    "3": [
      {
        "condition": "=H:=1:::::",
        "translation": "{Team2}",
        "shortTranslation": "W2"
      },
      {
        "condition": "::::::",
        "translation": "{Team2}",
        "shortTranslation": "W2"
      }
    ]
  },
  "periods": {
    "HB": [
      {
        "condition": "=2::=1",
        "translation": "2nd half"
      },
      {
        "condition": "=1::=1",
        "translation": "1st half"
      }
    ]
  },
  "tradingTypes": {
    "1": {
      "translation": "Goals"
```

```
    }
  }
}
```

Response contains four entities that are used in localization of markets:

- markets
- outcomes
- periods
- trading types

Each entity is a map of entity id to an array of entity translations.

All entities (except trading types) have complex rules for translation. These rules are compactly encoded in **condition** property.

Client code should check every condition in array and first matching should be used for translation.

Condition format

Condition consists of several parts separated by colon.

Number and meaning of condition parts is unique to each entity.

Each condition part is either simple expression (described below) or not set.

Condition part operators:

Operator	Meaning
=	value should be equal to provided constant
!	value should not equal to provided constant
>	value should be greater than provided constant
<	value should be less than provided constant

Entities condition format

Entity	Format
market	sportCondition:tradingTypeCondition:periodCondition:value1Condition:marketParameter2 sportCondition: example **=H:::** Condition is matched if sport (event schema) is Hockey (=H) tradingTypeCondition: example **:=1:::** Condition is matched if trading type is 1 (=1) (market schema, tradingType property from key) periodCondition: example **::=2::** Condition is matched if period is 2 (=2) (market schema, period property from key) value1Condition: example , **:::>4:** Condition is matched market value greater than 4 (>4) (market

	schema=>key=>marketParameters[0]) marketParameter2: example , **::::>4** Condition is matched market value greater than 4 (>4) (market schema=>key=>marketParameters[1])
outcome	sportCondition:marketTypeCondition:marketValue1Condition:outcomeValue1Condition:outcomeValuesCountCondition sportCondition: example **=H::::** Condition is matched if sport (event schema) is Hockey (=H) marketTypeCondition: example **:=1:::** Condition is matched if market type is 1 (=1) (market schema, market type property from key) marketValue1Condition: example **::>4":** Condition is matched market value greater than 4 (>4) (market schema=>key=>marketParameters[0]) outcomeValue1Condition: example , **:::=2:** Condition is matched if outcome parameter value is 2 (=2) (market schema=> outcomes => key => value[0]) outcomeValuesCountCondition: example: **::::=1** Condition is matched if outcome parameters count is 1 (=1) (market schema=> outcomes => key => value.Count())
outcome **v2**	sportCondition:marketTypeCondition:marketValue1Condition:marketValue2Condition:marketValue3Condition:outcomeValue1Condition:outcomeValuesCountCondition sportCondition: example **=H::::::** Condition is matched if sport (event schema) is Hockey (=H) marketTypeCondition: example **:=1:::::** Condition is matched if market type is 1 (=1) (market schema, market type property from key) **marketValue1Condition**: example **::>4"::::** Condition is matched market value greater than 4 (>4) (market schema=>key=>**marketParameters[0]**) **marketValue2Condition**: example **:::>4":::** Condition is matched market value greater than 4 (>4) (market schema=>key=>**marketParameters[1]**) **marketValue3Condition**: example **::::>4"::** Condition is matched market value greater than 4 (>4) (market schema=>key=>**marketParameters[2]**) outcomeValue1Condition: example , **:::::=2:** Condition is matched if outcome parameter value is 2 (=2) (market schema=> outcomes => key => value[0])

	outcomeValuesCountCondition: example: **::::::=1** Condition is matched if outcome parameters count is 1 (=1) (market schema=> outcomes => key => value.Count())
period	periodCondition:subPeriodCondition:periodsCountCondition periodCondition: example **=1::** Condition is matched if period is 1 (=1) (market schema => key => period) subPeriodCondition: example **:=1::** Condition is matched if subPeriod is 1 (=1) (market schema => key => subPeriod) periodsCountCondition: **::=1** periods count calculated if subPeriod not null than count = 2, else count = 1
tradingType	conditions are not used
prompt	sportCondition:tradingTypeCondition:periodCondition:subPeriodCondition sportCondition: example **=H:::** Condition is matched if sport (event schema) is Hockey (=H) tradingTypeCondition: example **:=1::** Condition is matched if trading type is 1 (=1) (market schema, tradingType property from key) periodCondition: example **::=2:** Condition is matched if period is 2 (=2) (market schema, period property from key) subPeriodCondition: example **:::=1** Condition is matched if subPeriod is 1 (=1) (market schema => key => subPeriod) Example "condition": "=CK:=1::!null", need to work with "null" for subPeriod

Translation format

Translation text could contain blocks that should be replaced by actual market (or outcome values).

Format	Replaced with	Example

{p1}	first market item value (with index 0)	translation: {
		• ru: "{p1}-й сет/ матч"
		}
		marketParameters [1]
		result
		translation: {
		• ru: "1-й сет/ матч"
		}

{p2}	second market item value (with index 1)	translation: {
		• ru: "Оба тайма тотал меньше {p2}"
		}
		marketParameters [1 1.5]
		translation: {
		• ru: "Оба тайма тотал меньше 1.5"
		}

{p3}	third market item value (with index 2)	translation: {
		• ru: "Больше {p3}"
		}
		marketParameters [1, 123213, 1.5]
		translation: {
		• ru: "Больше 1.5"
		}

{s1}	first outcome value (with index 0)	translation: {
		• ru: "Меньше {s1}"
		}
		outcome.key.values [0.5]
		translation: {
		• ru: "Меньше 0.5"
		}

{s2}	second outcome value (with index 1)	translation: {
		• ru: "С {s1} по {s2} минуту"
		}
		outcome.key.values [1 15]
		translation: {
		• ru: "С 1 по 15 минуту"
		}

#competitor{p1}	first market item value (with index 0) that will be translate by separate request by competitor	translation: {
		• ru: "#competitor{p1} - #competitor{p2}. Кто выше"
		}
		marketParameters [12323 1534234]
		translation: {
		• ru: "С 1 по 15 минуту"

		}

{Team{p1}}	first market item value (with index 0) that will be translate by separate request by team	translation: {
		• ru: "Индивидуальный тотал {Team{p1}}"
		}
		marketParameters [1000]
		request to strapi by team id = 1000, Team should be removed
		translation: {
		• ru: "Индивидуальный тотал Реал"
		}

#player{p1}	first market item value (with index 0) that will be translate by separate request by payer	translation: {
		• ru: "#player{p1}"
		}
		marketParameters [1000]
		request to strapi by player id = 1000, #player should be removed
		result
		translation: {
		• ru: "Рональду"
		}

#player{s1}	first outcome value (with index 0) that will be translate by separate request by payer	translation: {
		• sB_ru: "#player{s1} больше {p1}"
		}
		marketParameters [0.5]
		outcome.key.values [1000]
		request to strapi by player id = 1000, #player should be removed
		result
		translation: {
		• sB_ru: "Рональду больше 0.5"
		}

{Team{p1}}	**first market item value (with index 0) that will be translate as**	translation: { • ru: "Индивидуальный тотал {Team{p1}}"

	first or second team name in pair competitor1 - competitor2	}
		marketParameters [1]
		Event competitors: Реал - Барселона
		translation: {
		• ru: "Индивидуальный тотал Реал"
		}

{TeamID1}	Remove tag, take all markets parameters and translate it by separate request by team	translation: {
		• ru: "{TeamID1}Выйдет в 1/8 финала"
		}
		marketParameters [12323]
		translation: {
		• ru: "Выйдет в 1\8 финала"
		}

{r1}	**period** value	translation: {
		• ru: "После {r1}-й минуты"
		}
		period: 1
		subPeriod: 6000
		translation: {
		• ru: "После 1-й минуты"
		}

{r2}	**subPeriod** value	translation: {
		• ru: "С {r1}-й по {r2}-ю минуту включительно"
		}
		period: 1
		subPeriod: 15
		translation: {
		• ru: "С 1-й по 15-ю минуту включительно"
		}

After replacing values teams can be replaced:

Format	Replaced with
{Team1}	First team name
{Team2}	Second team name

Examples

Markets:

```
{
    "condition": "=T:::>4:", // Condition is matched if sport is Tennis (=T) and market value greater than 4 (>4)
    "translation": "Race to {p1}" // {p1} should be replaced with actual market value
}
```

Outcomes 1.0:

```
{
    "condition": "=H:=1:::", // Condition is matched if sport is Hockey (=H) and market type is 1 (=1)
    "translation": "{Team2}" // {Team1} should be replaces with first team name
}
```

Outcomes 1.1 (added short translation):

```
{
    "condition": "=H:=1:::", // Condition is matched if sport is Hockey (=H) and market type is 1 (=1)
    "translation": "{Team2}", // {Team1} should be replaces with first team name
    "shortTranslation": "W2"
},
{
    "condition": "::::", // Condition is matched in any case
    "translation": "Over {p1}", // {p1} should be replaced with actual market value
    "shortTranslation": "Over {p1}" // {p1} should be replaced with actual market value
},
```

Periods:

```
{
    "condition": "=1::=1", // Condition is matched if period is 1 (=1) and number of periods is 1 (=1)
    "translation": "1st period"
}
```

Javascript sample implementation

Localization format is made in such a way that is should be parsed in client code easily.

Below is implementation in JavaScript:

```
const MarketLocalization = {

    cache: {},

```

```
    // Load localization info once
    async loadCache() {
        let response = await fetch("/api/localization/markets?lang=en");
        this.cache = await response.json();
    },

    // Find market translations by marketType then iterate over conditions.
    getMarketTranslation(marketType, sport, tradingType, period, marketItemValue) {
        let item = (this.cache.markets[marketType] || [])
            .find(item => this.isMatch(item.condition, [sport, tradingType, period, marketItemValue]));
        return item ? item.translation : "";
    },

    // Find outcome translations by outcomeType then iterate over conditions.
    getOutcomeTranslation(outcomeType, sport, marketType, marketValue, outcomeValue1, outcomeValuesCount) {
        let item = (this.cache.outcomes[outcomeType] || [])
            .find(item => this.isMatch(item.condition, [sport, marketType, marketValue, outcomeValue1, outcomeVa
        return item ? item.translation : "";
    },

    // Find period translations by sport then iterate over conditions.
    getPeriodTranslation(sport, period, subPeriod) {
        let item = (this.cache.periods[sport] || [])
            .find(item => this.isMatch(item.condition, [period, subPeriod, subPeriod ? 2 : 1]));
        return item ? item.translation : "";
    },

    // Find trading type translation by tradingType (without iteration).
    getTradingTypeTranslation(tradingType) {
        let item = this.cache.tradingTypes[tradingType];
        return item ? item.translation : "";
    },

    // Simple parsing of condition expressions, in weakly typed languages (like JS) works for both integers and
    isMatch(conditionExpression, values) {
        return conditionExpression.split(":").every((condition, i) => {
            switch ((condition || '*')[0]) {
                case '*': return true;
                case '!': return values[i] != condition.substring(1);
                case '>': return values[i] > condition.substring(1);
                case '<': return values[i] < condition.substring(1);
                case '=': return values[i] == condition.substring(1);
            }
        });
    },

    // Regex to replace market values.
    replaceMarketValues(text, values) {
        return text.replace(/{p(\d)}/g, (match, g1) => values ? values[parseInt(g1) - 1] : match);
    },

    // Regex to replace outcome values.
    replaceOutcomeValues(text, values) {
        return text.replace(/{s(\d)}/g, (match, g1) => values ? values[parseInt(g1) - 1] : match);
    }
}
```