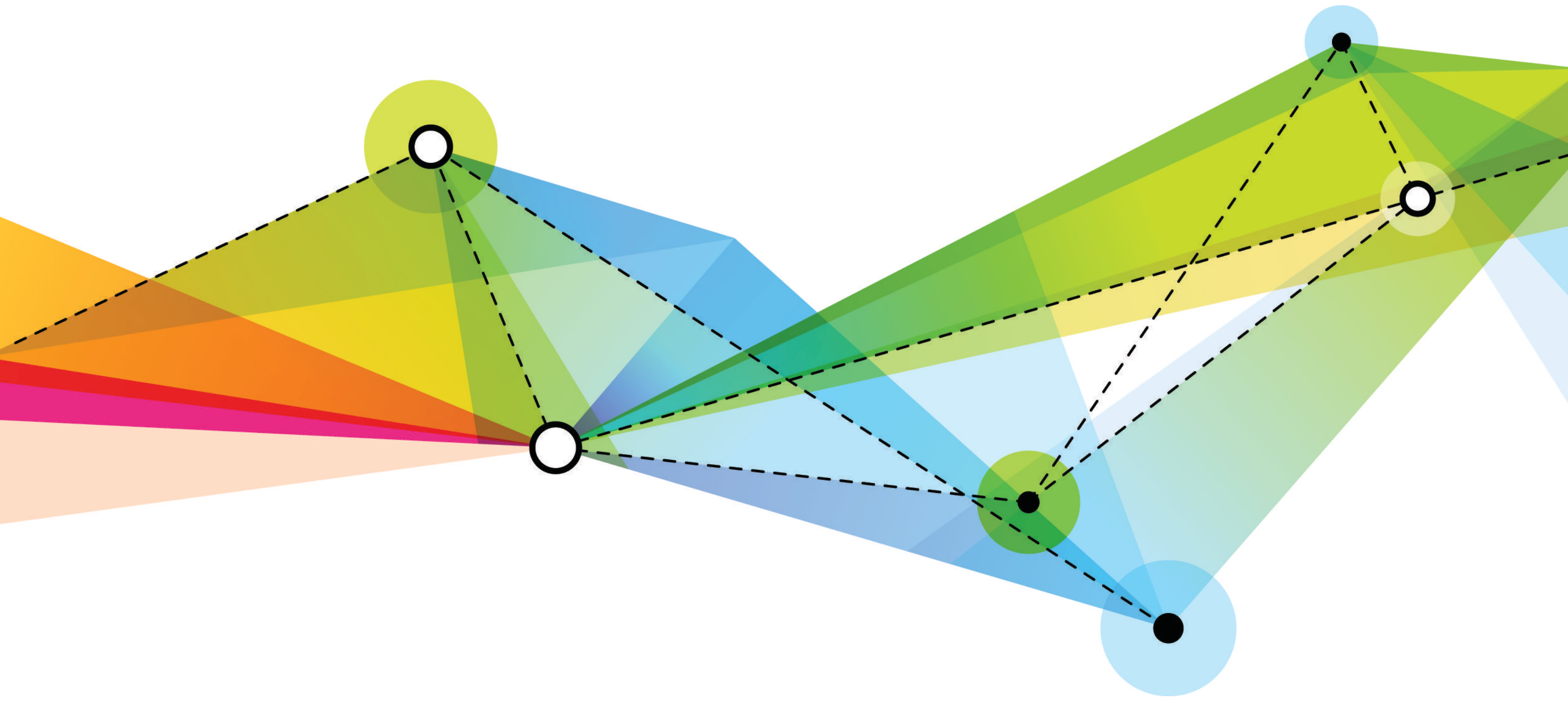
inomial

Flexible Rate Card Importer



Edition: 1.0
Release date: August 23, 2016
Smile version: 6.0

Published by Inomial Pty Ltd
Suite 801, 620 Bourke St, Melbourne, Vic 3000, Australia
www.inomial.com • +61 3 9663 3554 • sales@inomial.com • support@inomial.com

Flexible Rate Card Importer

Introduction

The flexible rate card importer provides a means for importing an entire rate card, including prefixes, tariffs and destinations into Smile, via a CSV file. Any prefixes, tariffs or destinations that do not already exist will be created.

The flexible rate card importer uses JExpr to express how Smile should handle an imported CSV column. For more information, see the *JExpr Language Guide*.

This document describes how to configure a Smile flexible rate card importer to process a CSV rate card import file.

Note: JExpr is intended for advanced users familiar with programming languages. Contact Inomial for assistance.

Rate card template and importer

Rate card importers are used for importing rate card details into Smile against a relevant subscription.

An appropriate rate card template must be configured before a rate card can be imported for a subscription. A rate card template specifies the configuration options applied during import and any error actions.

Add a rate card template

Smile rate card templates define how Smile processes the rate card details from the CSV import file.

This task explains how to add a rate card template.

1. Select **Rate card upload templates** under **Services, Ordering & Rating** on the Configuration and Tools page.

   The Rate Card Template page is displayed.
2. Click **Add...**.

   A Rate Card Template Configuration page is displayed.
3. Type a name for the rate card in the **Template name** field.
4. Complete the required rate card template configuration fields.

   For more information, see Import template field properties and JExpr dictionaries.
5. Click **Save**.

   The Rate Card Template page is displayed. The template is added to the **Rate Card Templates** summary list.

Import a rate card

Rate cards are imported at the subscription level in Smile. A rate card template is required to import a rate card CSV file.

Note: Not all services or subscriptions can see the **Rate cards** sub-menu and import rate cards. A service's **Account Configuration Form** entry must be configured with *svcVoiceSettlement* to enable this function. To view service configuration select **Service Types** under **Services, Ordering & Rating** on the Configuration and Tools page.

This task explains how to import a rate CSV file to a subscription.

1. Search for the subscription to which you want to add a rate card, then double-click the subscription entry in the search results.

   The subscription's summary page is displayed.
2. On the Account Menu Tree, click the ▶ to the left of the subscription entry to view sub-menu entries. Click **Rate cards**.

   The Rate Cards page is displayed.
3. Click **Add...**

   The **Plan Schedule Item Configuration** window is displayed.
4. Complete the required options. The following fields require additional explanation.
   - **Plan**—Specifies the plan the rate card applies to. Select a plan from the **Plan** drop-down.
   - **Start**—(Optional) Specifies the date from which to apply the rate card. The current date is displayed by default. Click the 📅 to select a different date/time.
   - **Replace future plans?**—(Optional) When selected, specifies if the plan/rate card replaces any future-dated plan configurations.
5. Click **Continue...**

   The **Upload Rate Card** window is displayed.
6. Complete the required fields. The following fields require additional explanation.
   - **Template**—Specifies the rate card template to use to process the import file. Select a template from the **Template** drop-down. For more information, see Add a rate card template.
   - **File**—Specifies the CSV file to import. Click **Select file** to navigate to your import file or drag and drop your file onto this window.
7. Click **Upload**.

   If no errors are detected during processing, the rate card is displayed.
8. Click **Submit**.

   The rate card is submitted and displayed in the **Plan Schedule Items** table.

Import template field properties

The following fields are used to configure the JExpr properties of a flexible rate card template.

Skip Expression

Values: true, false

Purpose: (Optional) Defines an expression that if true will cause the importer to skip the specified entry.

Example:

```
lineNumber <= 1
```

The header line will be skipped.

Invalidation Expression

Values: true, false

Purpose: (Optional) Defines an expression that if true will determine that the CSV file is invalid and cause the import to be aborted.

Example:

```
$$ < 15
```

The import will abort if any line has less than 15 columns.

Destination Name

Format: String

Purpose: Specifies the name of the destination that the tariff represents.

Example:

```
$1
```

The destination name is the first column.

Destination Prefix

Format: String

Purpose: Specifies the prefix of numbers that a CDR must have to use this tariff.

Example:

```
$2
```

The destination prefix is the second column.

Tariff Type Name

Format: String

Purpose: Specifies the name of the tariff type to use for this tariff. If this tariff type does not exist the import will be aborted.

Example:

```
$3
```

The tariff type name is the third column.

Tariff Flag Fall

Format: BigDecimal

Purpose: Specifies the flag fall amount to use for this tariff.

Example:

```
$4::BigDecimal
```

The tariff flag fall is the fourth column.

Tariff Rate per Charging Unit

Format: BigDecimal

Purpose: Specifies the rate per charging unit to use for this tariff.

Example:

```
$5::BigDecimal
```

The tariff rate per charging unit is the fifth column.

Tariff Charging Unit Size

Format: Integer

Purpose: Specifies the charging unit size to use for this tariff.

Example:

```
60
```

Indicates that the Tariff Rate per Charging Unit is a per minute rate.

Tariff Charging Increment

Format: Integer

Purpose: Specifies the charging increment to use for this tariff.

Example:

```
30
```

Indicates that calls are charged in 30 second increments.

Tariff Minimum Charge

Format: BigDecimal

Purpose: (Optional) Specifies the minimum charge to use for this tariff.

Example:

```
$6::BigDecimal
```

The tariff minimum charge is the sixth column.

Tariff Maximum Charge

Format: BigDecimal

Purpose: (Optional) Specifies the maximum charge to use for this tariff.

Example:

```
$7::BigDecimal
```

The tariff maximum charge is the seventh column.

Tariff Call Type

Format: CallType

Purpose: Specifies the call type to use for this tariff.

Example:

```
CallType.getVoice()
```

The tariff is categorised as ordinary voice call type.

Tariff Decimal Places

Format: Integer

Purpose: Specifies the number of decimal places to use for this tariff.

Example:

```
6
```

The tariff charge is displayed to 6 decimal places.

Tariff Rounding Mode

Format: RoundingType

Purpose: Specifies the roundingType to use for this tariff.

Example:

```
RoundingType.getHalfUp()
```

The tariff is rounded to it's nearest neighbour.

JExpr dictionaries

The JExpr language can feature pre-defined dictionaries for certain parts of Smile that accept JExpr expressions for their configuration.

Dictionaries are a list of pre-defined functions. These functions can take zero or more arguments, each argument being a JExpr expression itself. The function will return an expression of a particular JExpr type (or may return null).

A function call in JExpr looks like the following:

```
parsers.parseDate("yyyy-MM-dd", $1);
```

In this example:

- `parsers` is the dictionary namespace that contains the function
- `parseDate` is the name of the function
- This function takes two arguments (both of string type), and returns a Java date/timestamp (java.util.Date) value
- Parentheses encapsulate all of the arguments
- Commas separate individual arguments

parsers dictionary

This dictionary contains functions that direct how specified strings should be parsed.

`parseDate()`

```
parsers.parseDate(format, sourceText)
```

Parses a string containing a date and/or timestamp specification in a particular structured format.

Parameters

format

Format: string

Purpose: Describes the expected layout of the individual date/time components within the sourceText argument.

The format string can contain the following (case-sensitive) placeholders:

yy	Two-digit year. For example, `99` for 1999.
yyyy	Four-digit year. For example, `1999`.
MMMM	Full month name. For example, `July`.
MMM	Three-letter month abbreviation. For example, `Jul` for July.
MM	Two-digit numeric month. For example, `07` for July.
dd	Two-digit day in month. For example, `23`.
HH	Two-digit hour-in-day. For example, `00` thru `23` inclusive, for 24-hour time.
hh	Two-digit hour-in-day. For example, `01` thru `12` inclusive, for 12-hour time.
mm	Two-digit minute in hour. For example, `00` thru `59` inclusive.
ss	Two-digit second in minute. For example, `00` thru `59` inclusive.
SSS	Three-digit millisecond value. For example, `000` thru `999` inclusive.
z	Time zone. For example, `UTC+10:00`.
Z	RFC 822 time zone. For example, `+1000` for 10 hours ahead of UTC.
a	AM/PM marker. For example, `AM` or `PM`.

To escape a letter (indicate that it needs to be matched verbatim), enclose it in single quotes `'`. To match a single quote directly, write two single quotes consecutively `''` (no space between quotes).

Note: All the sequences supported by the Java Foundation Class java.text.SimpleDateFormat are permitted for this parameter. For more information, see http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html

The letters 'G', 'Y', 'L', 'W', 'w', 'D', 'F', 'E', 'u', 'k', 'K', 'X' are also supported as per the above Java specification.

Any other characters, such as punctuation, spaces or letters not mentioned above, will be matched verbatim.

sourceText

Format: string

Purpose: Specifies the date string to be parsed.

Example: A parseDate request

The following example is a `parseDate` request containing a string for the `format` argument and `sourceText` of the date to parse.

```
parsers.parseDate("dd/MM/yyyy HH:mm:ss", "31/12/1999 23:59:59")
```

Result

Returns a java.util.Date instance containing the parsed date/timestamp.

Example: A returned parseDate request

This example shows the returned parseDate request.

```
31st December 1999 at 11:59:59pm
```

CallType dictionary

All the functions in this dictionary return a specific value from a set of enumerated values that can be used to categorise a type of CDR record.

None of these functions take any arguments.

`getVoice()`

```
CallType.getVoice()
```

Returns ordinary voice call values.

`getData()`

```
CallType.getData()
```

Returns data session values.

`getSms()`

```
CallType.getSms()
```

Returns SMS text message values.

`getMms()`

```
CallType.getMms()
```

Returns MMS message values.

`getFax()`

```
CallType.getFax()
```

Returns fax call values.

`getWap()`

```
CallType.getWap()
```

Returns WAP session values.

`getForwardedVoice()`

```
CallType.getForwardedVoice()
```

Returns forwarded voice call values.

`getCount()`

```
CallType.getCount()
```

Returns CDR total count record values.

`getUnknown()`

```
CallType.getUnknown()
```

Returns unknown call type values.

`getVideoCall()`

```
CallType.getVideoCall()
```

Returns video call values.

`getIsdn()`

```
CallType.getIsdn()
```

Returns ISDN session values.

`getImportedCharge()`

```
CallType.getImportedCharge()
```

Returns imported charge (from external system) values.

RoundingType dictionary

All the functions in this dictionary enumerate a type of rounding that can be applied to a tariff charge.

None of these functions take any arguments.

`getNone()`

```
RoundingType.getNone()
```

Specifies that no rounding is applied. An error will be triggered if rounding is required to reach specified decimal places.

`getUp()`

```
RoundingType.getUp()
```

Specifies to always round up (away from zero).

`getDown()`

```
RoundingType.getDown()
```

Specifies to always round down (towards zero).

`getHalfUp()`

```
RoundingType.getHalfUp()
```

Specifies to round towards "nearest neighbour". If both neighbours are equidistant then round up.

`getHalfEven()`

```
RoundingType.getHalfEven()
```

Specifies to round towards "nearest neighbour". If both neighbours are equidistant then round towards the nearest even neighbour

lineNumber

```
lineNumber
```

lineNumber provides the line number currently being processed (where 1 is the first line).

Example:

```
lineNumber <= 1
```

When used in a skip expression, the header line is skipped.

$$

```
$$
```

Specifies the number of cells on the current line.

Example:

```
$$ < 15
```

If used in an invalidation expression the import will abort if any line has less than 15 columns.

Example:

```
$$ == 4
```

There must be exactly 4 columns for an item row.

$x

```
$x
```

Specifies the *x*th cell in the current line (where $1 is the first cell).

Example:

```
$2
```

The second column.

Example:

```
$4::BigDecimal
```

The fourth column, formatted as BigDecimal.

Example:

```
$17::UTInteger
```

The seventeenth column, formatted as UTInteger.