Internationalization

Version 2018

Contents

Internationalization

Introduction

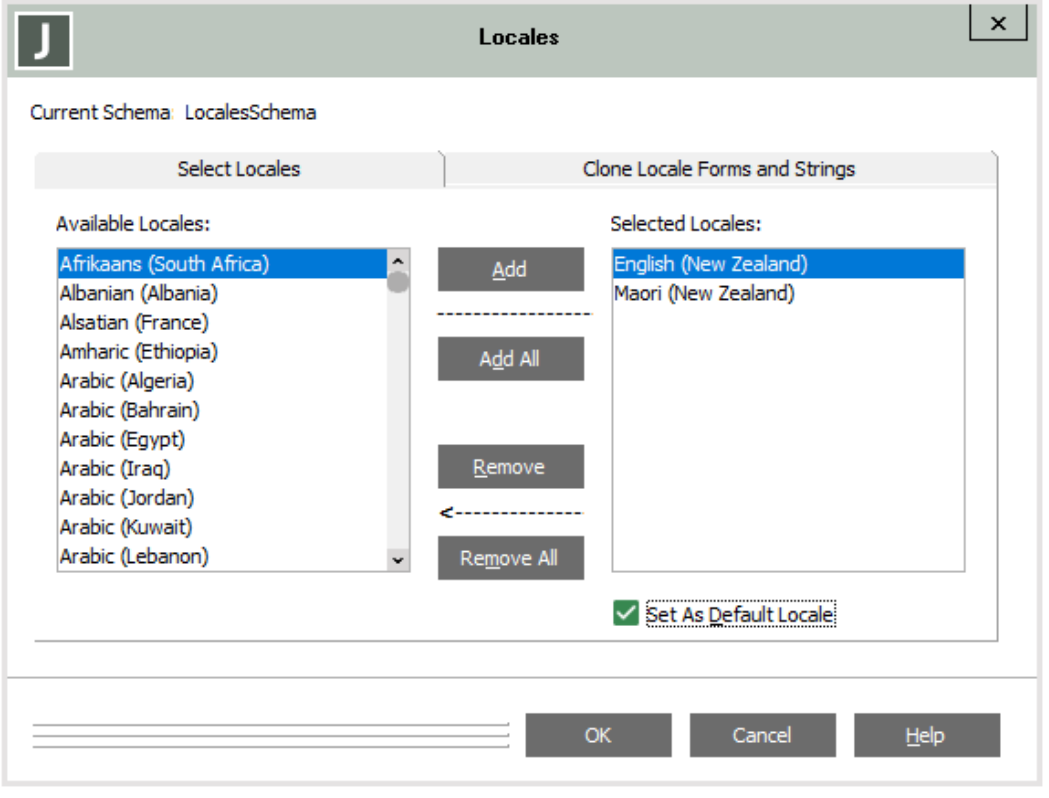
JADE provides for the ability to internationalize JADE systems to be compatible with a wide variety of languages and formats. By using translatable strings, you can specify translations for any text in any number of different languages.

The Formats Browser allows for the specification of date, currency, and number formats (which may differ, between locales).

Locales

In JADE, a locale refers to a language and the formatting considerations associated with the use of that language. For example, while the language of **English (New Zealand)** and **English (United States)** is similar (other than a few minor spelling differences), a New Zealand user will likely be used to a date format of **dd/mm/yyyy** while an American user might prefer **mm/dd/yyyy**. Having separate locales for New Zealand and the United States therefore could allow for all dates to be presented in the appropriate format for each country.

Use the **Locales** command in the Schema menu to specify which locales are to be supported in a specific schema.



When running a JADE application, forms are displayed in the default locale specified by the operating system. You can use the **Application** class **setJadeLocale** method to explicitly set the locale; for example, on a form that allows the user to select their language from the form itself. To use this method, you must provide it with the Language Code Identifier (LCID) number of the target locale.

To find the appropriate LCID, each locale has a unique LCID number and a corresponding global constant. For example, the New Zealand English locale has the LCID 5129 and the **JadeLocaleIdNumbers** category global constant **LCID_English_NewZealand**.

Locales in Multiuser Systems

At run time, JADE applications automatically select the appropriate locale based on the client's operating system. For single user and fat client systems, the presentation locale always matches the application locale. However, more consideration must be taken when JADE is running in thin client mode.

The **EnhancedLocaleSupport** parameter in the [JadeEnvironment] section of the JADE initialization file determines whether the locale settings are synchronized between the standard client (application server) and the thin client.

When the **EnhancedLocaleSupport** parameter is not defined or it is set to **false**:

- Application server regional overrides are suppressed on the thin client; that is, the **Application** class **setJadeLocale** method no longer has any effect on the locale for the thin client.
- The two-digit mask year of **yy** is calculated using the current century. For example, **99** becomes **2099** rather than **1999**.

When the **EnhancedLocaleSupport** parameter is set to **true**:

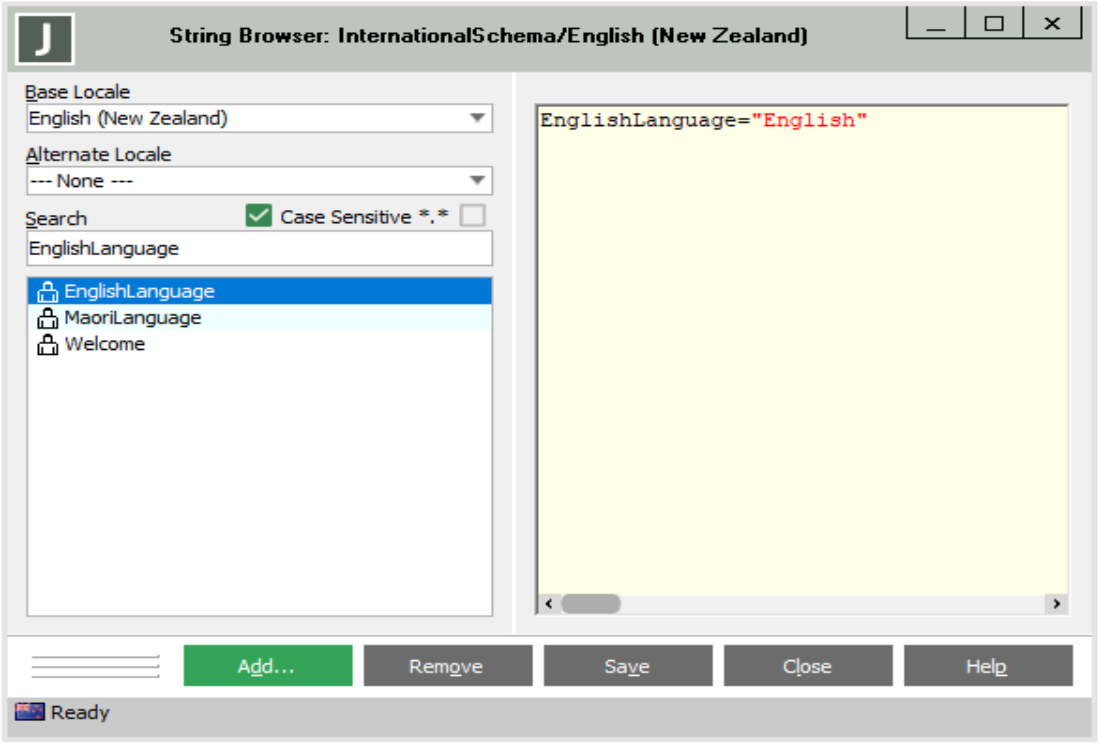
- The required language or languages must be installed on both the application server and the thin client.
- Regional overrides are applied; that is, the **Application** class **setJadeLocale** method overrides the operating system locale on the thin client.
- The Windows Control Panel setting is used to convert a two-digit year into a four-digit year for a two-digit edit mask year of **yy**. For example, **99** by default becomes **1999** rather than **2099**.

Translatable Strings

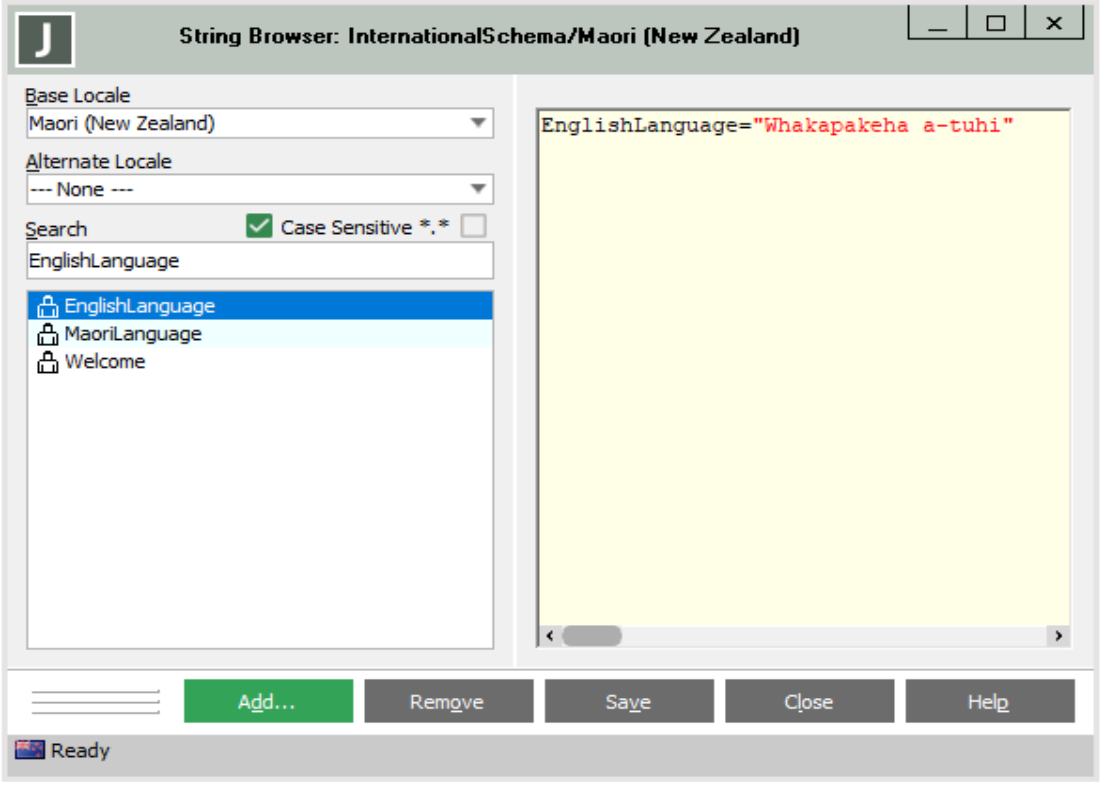
Translatable strings are like global constant strings, but with the important distinction that multiple values can be assigned to the same translatable string: one for each of the locales supported by the schema.

By diligently using translatable strings rather than string literals, it becomes comparatively easy to translate JADE applications between languages and provides the possibility of having a JADE application automatically translate to the user's language based on his or her locale settings.

Translatable strings can be added to a schema from the String Browser by clicking the **Add** button, and defined for each locale by selecting the appropriate locale from the **Base Locale** combo box and then modifying the definition for that locale.



The following example shows the translation from the **EnglishLanguage** translatable string.



When a translatable string has been defined in a schema, it can be added to any JADE method within that schema in place of a string literal, by prefixing the name of the translatable string with a dollar sign (**$**).

When the locale is set to **English (New Zealand)**, the first of the following examples is the same as the second example.

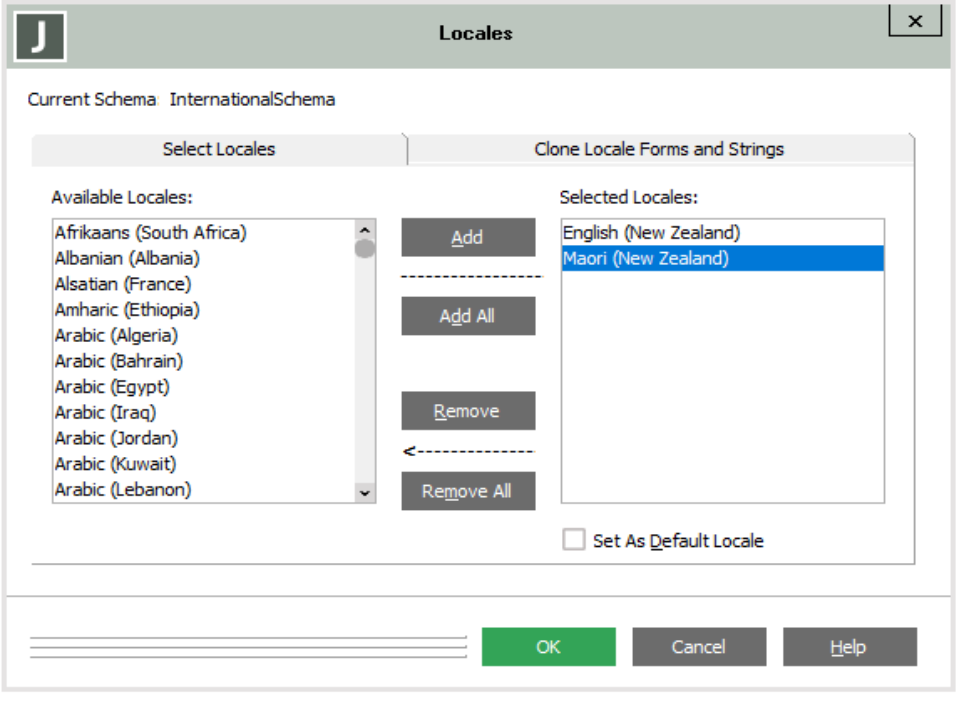
```
translatableStringExample();

begin
    write "A language of New Zealand is " & $EnglishLanguage;
end;
```

```
translatableStringExample();

begin
    write "A language of New Zealand is " & "English";
end;
```

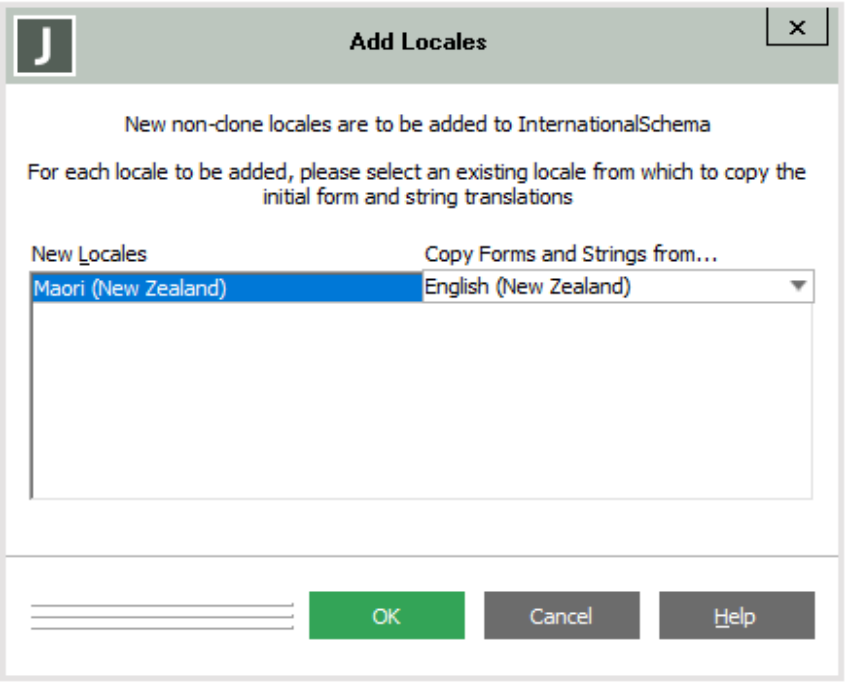
Exercise 1 – Addling a Locale

In this exercise, you will create a new schema and add a locale for te reo Māori (that is, the Maori language) alongside New Zealand English. You will also design a simple form that is translated to and from te reo Maori in future exercises.

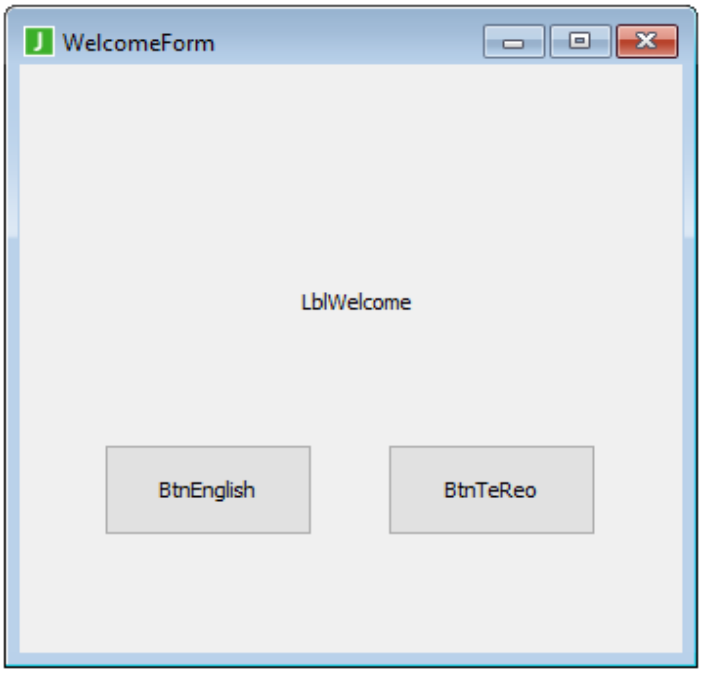
1. Create a schema called **InternationalSchema**.
2. When **InternationalSchema** is selected in the Schema Browser, select the **Locales** command from the Schema menu.
3. From the **Available Locales** list box, select **Maori (New Zealand)**, add it to the **Selected Locales** list box, and then click **OK**.



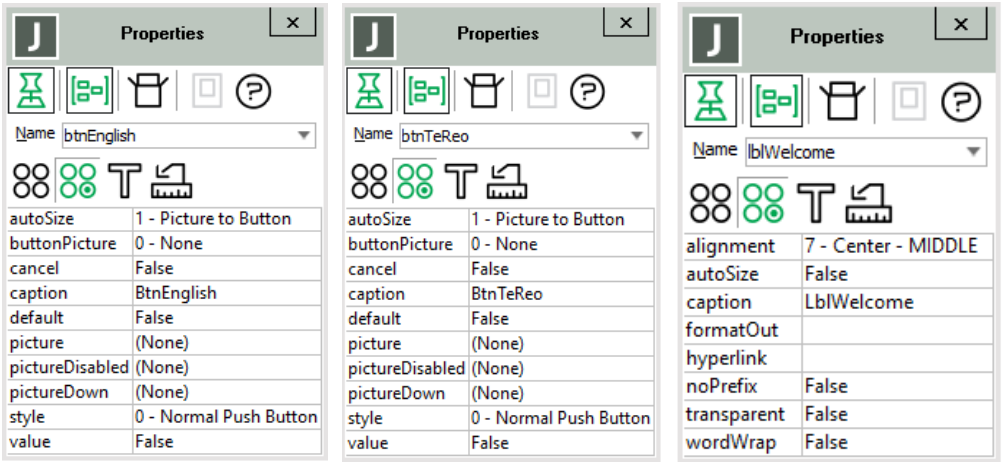
4. On the Add Locales dialog that is then displayed, click **OK**.



5. Open the JADE Painter and create a form called **WelcomeForm**.



6. Design the WelcomeForm as follows.



Note The control captions don't matter, as we will be setting these programmatically (that is, from the JADE code).

7. Save the form.

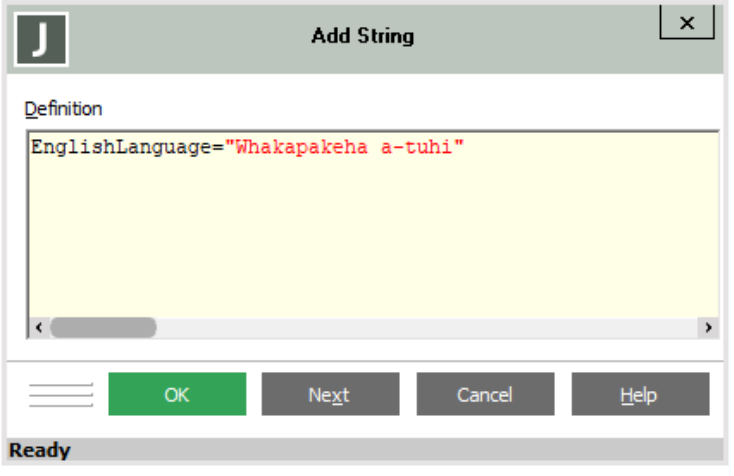
Exercise 2 – Adding and Using Translatable Strings

In this exercise, you will create translatable strings with two definitions for each string: one for English and one for te reo Maori.

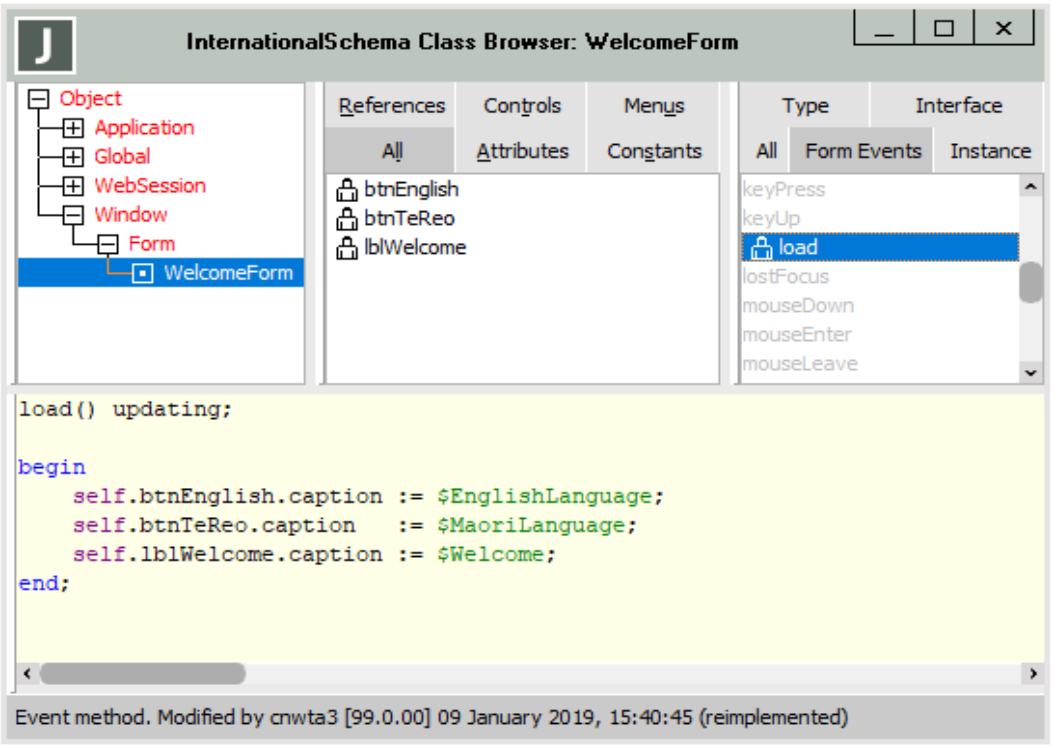
1. Select **InternationalSchema** in the Schema Browser, select the **Strings** command from the Schema menu, and then add the following translatable strings.

String Name	English	Maori
EnglishLanguage	English	Whakapakeha a-tuhi
MaoriLanguage	Maori	Te Reo
Welcome	Welcome and thank you for visiting.	Nau mai, ka mihi ki a koe mo te haerenga.

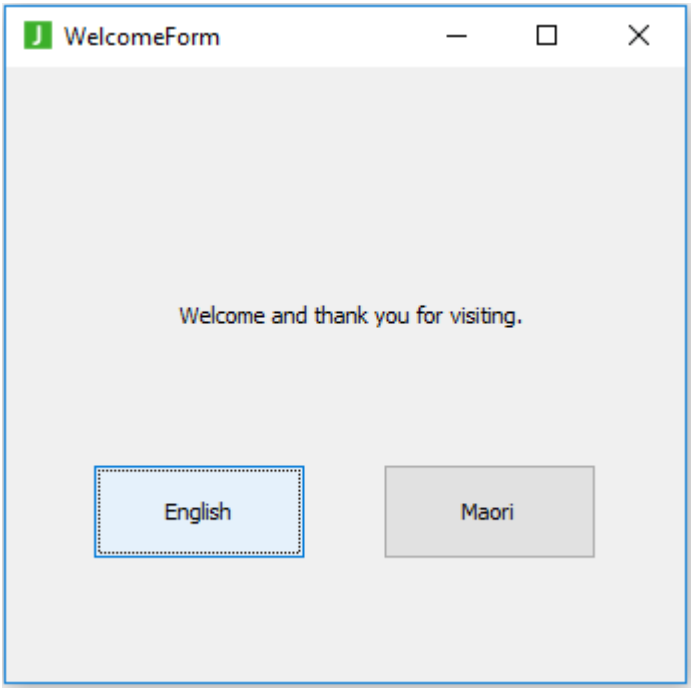
To add a new string, click **Add** in the String Browser and then specify it in the format *string-name*="*description*"; for example:



2. In the **Form Events** folder for the **WelcomeForm**, code the **load** method as follows.



3. Run the **InternationalSchema** application by right-clicking on the **Run Application** toolbar button. You should see the following form, in English.



Exercise 3 – Translating Translatable Strings

In this exercise, you will allow for the runtime translation of the Welcome Form between English and Maori by using the the **Application** class **setJadeLocale** method.

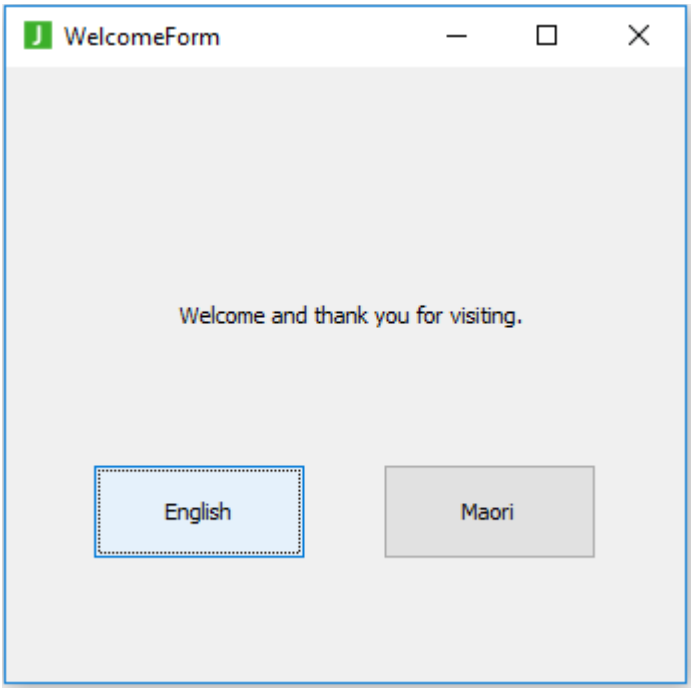
1. Code the **click** method of the **btnEnglish** button as follows.

```
btnEnglish_click(btn: Button input) updating;

begin
    app.setJadeLocale(LCID_English_NewZealand);
    self.load();
end;
```

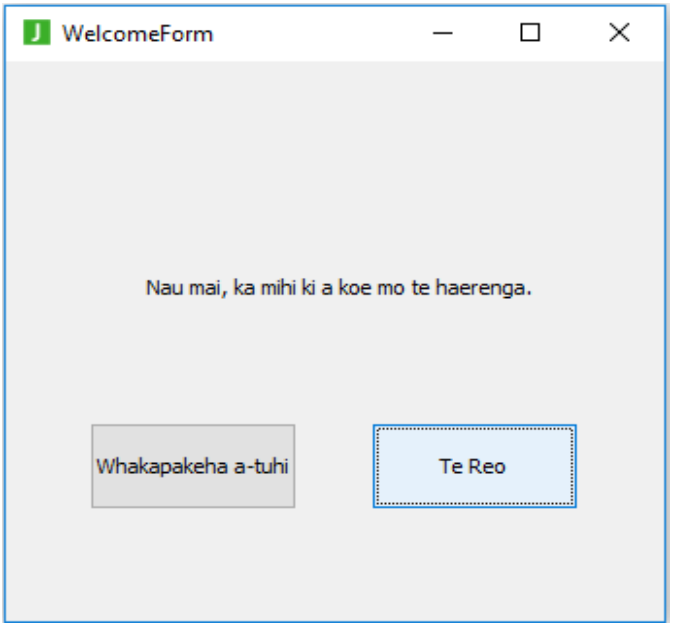
2. Code the **click** method of the **btnTeReo** button as follows.

```
btnTeReo_click(btn: Button input) updating;

begin
    app.setJadeLocale(LCID_Maori_NewZealand);
    self.load();
end;
```

3. Run the **InternationalSchema** application by right-clicking on the **Run Application** toolbar button. You should see the following form, in English.



4. Click the **Maori** button, to translate the form into te reo Maori.



5. Click the **Whakapakeha a-tuhi** button, to translate the form back into English.

Note Typically, usability best practices suggest that all controls that translate a page or form should be written in the target language rather than the currently selected language. However, this example shows the translation of strings rather than being an example of good usability.

Programmatically Maintaining Translatable Strings

In addition to using the String Browser to maintain translatable strings, the **Schema** class **addCompileTranslatableString** and the **TranslatableString** class **updateCompile** methods enable you to add and edit translatable strings, respectively.

The **addCompileTranslatableString** method, which is used to add new translatable strings to all locales of a schema at run time, takes the following parameters.

Parameter	Type	Purpose
Source	String	The translatable string definition to be added to the schema and must be in the format *string-name="description"*. However, as nesting " " characters is not possible if using a string literal, single quote ' ' characters can be substituted, as required.
errorCode	Integer output	This output parameter is set to the error code after running the method. It is zero (**0**) if there is no error or it can be turned into a descriptive error message by using the **Process** class **getErrorText** method.
errorOffset	Integer output	This output parameter is set to the position in the **source** parameter where the error was encountered. Note that unlike most JADE strings and arrays, this offset begins at zero (**0**) for the first character of the source string rather than at **1**.
errorLength	Integer output	This output parameter is set to the number of characters in the **source** parameter that were in error.

The **updateCompile** method is to modify existing translatable strings and it takes the same parameters as the **addCompileTranslatableString** method. However, as this method is called from a translatable string rather than a schema, the target translatable string must first be located, and changes are specific to a target locale within that schema.

For example, to find and modify the **Welcome** string for the **English (New Zealand)** locale, you must first find the **English (New Zealand)** locale using the **Schema** class **getLocale** method, then retrieve the translatable string using the **getTranslatableStringLocal** method of the located **Locale**.

Exercise 4 – Adding a Translatable String Programmatically

In this exercise, you will use the **Schema** class **addCompileTranslatableString** method to add a translatable string to all base locales of the **InternationalSchema** schema.

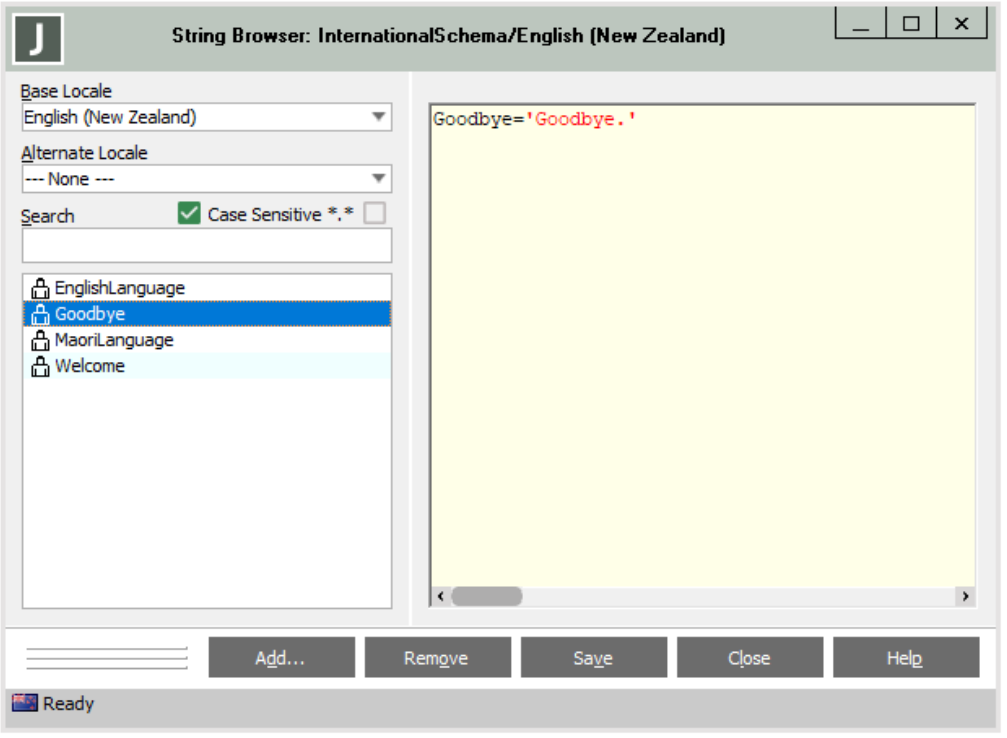
1. Add a JadeScript method called **addTranslatableString** and code it as follows.

```
addTranslatableString();

vars
    errorCode, errorOffset, errorLength : Integer;
begin
    beginTransaction;
    currentSchema.addCompileTranslatableString(
                                        "Goodbye='Goodbye.'",
                                        errorCode,
                                        errorOffset,
                                        errorLength
                                        );
    if errorCode = 0 then
        commitTransaction;
    else
        write "ERROR: " & process.getErrorText(errorCode);
        write "At position " & errorOffset.String;
        write errorLength.String & " characters were invalid";
    endif;
end;
```

Note If there is an error, the **addCompileTranslatableString** method automatically aborts the transaction, so we don't need to include an **abortTransaction** instruction.

2. Run the method and then open the String Browser.

You should see that the **Goodbye** string has been added to the list of translatable strings.



However, viewing the **Maori (New Zealand)** locale shows that the same value has been given for the English and Maori languages.

Exercise 5 – Updating a Translatable String Programmatically

In this exercise, you will use the **TranslatableString** class **updateCompile** method to set the **Maori (Te Reo)** definition for the **Goodbye** string.

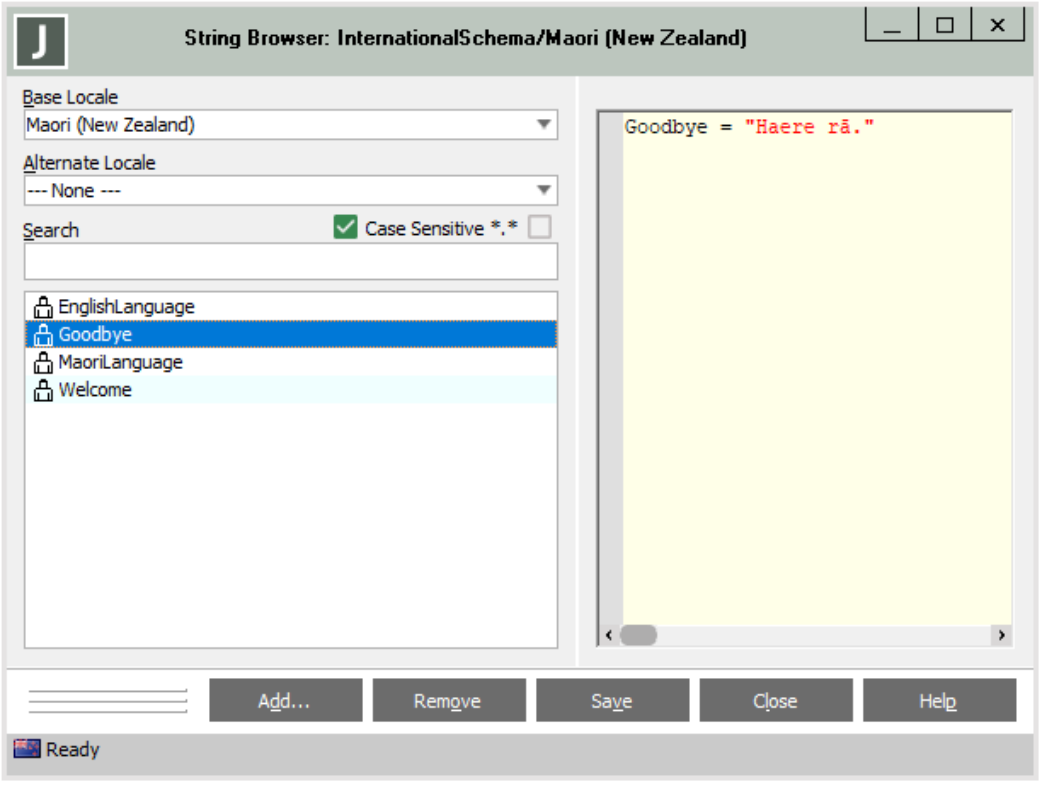
1. Add a JadeScript method called **updateTranslatableString** and code it as follows.

```
updateTranslatableString();

vars
    locale : Locale;
    errorCode, errorOffset, errorLength : Integer;
begin
    locale := currentSchema.getLocaleLocal(LCID_Maori_NewZealand.String);

    beginTransaction;
    locale.getTranslatableStringLocal("Goodbye").updateCompile(
                                                    'Goodbye = "Haere rā."',
                                                    errorCode,
                                                    errorOffset,
                                                    errorLength);

    if errorCode = 0 then
        commitTransaction;
    else
        write "ERROR: " & process.getErrorText(errorCode);
        write "At position " & errorOffset.String;
        write errorLength.String & " characters were invalid";
    endif;
end;
```

Note In ANSI JADE systems, the **ā** character is replaced by an **a**.

2. Run the method and then open the String Browser.

   You should see that the **Goodbye** string's definition has been changed to **Haere rā** for the Maori locale.



Date Formats

By default, when using the **shortFormat** or **longFormat** method of the **Date** primitive type, JADE uses the date format of the current locale.

However, you can explicitly specify a date format for a specific date string by:

- Using the **format** method of the **Date** type.

  This method takes a string representation of a date format and returns the date in the specified format. For example, passing "**dd.MM.yyyy**" returns **22.01.2019** if the date is the 22nd of January 2019.

- Creating user date formats with the Format Browser and then using the **userFormat** method of the **Date** primitive type.

The string representation of the date format that is passed to the **format** method of the **Date** primitive type is called the *picture* of the format.

The picture can contain the following string picture elements, or tokens. (Separate each element with a space or separator character.)

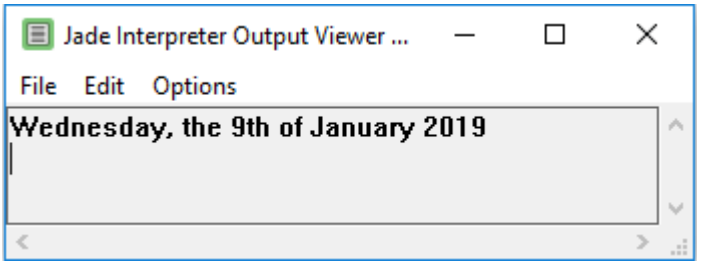
Picture	Description	Output Example
d	Day of the month as digits, with no leading zero.	9
dd	Day of the month as digits, with a leading zero, if applicable.	09
ddd	Day of the month as abbreviated name, as specified in the locale definition.	Wed
dddd	Day of the week as the full name.	Wednesday
M	Month as digits, with no leading zero.	1
MM	Month as digits, with a leading zero, if applicable.	01
MMM	Month as abbreviated name, as specified in the locale definition.	Jan
MMMM	Month as full name.	January
y	Year, represented by only the last two digits, or last one digit if the last two digits are less than 10.	8
yy	Year, represented by only the last two digits.	18
yyy	Year, represented by all significant digits.	2018

Note The locale definitions for dates are specified by the operating system, rather than in JADE.

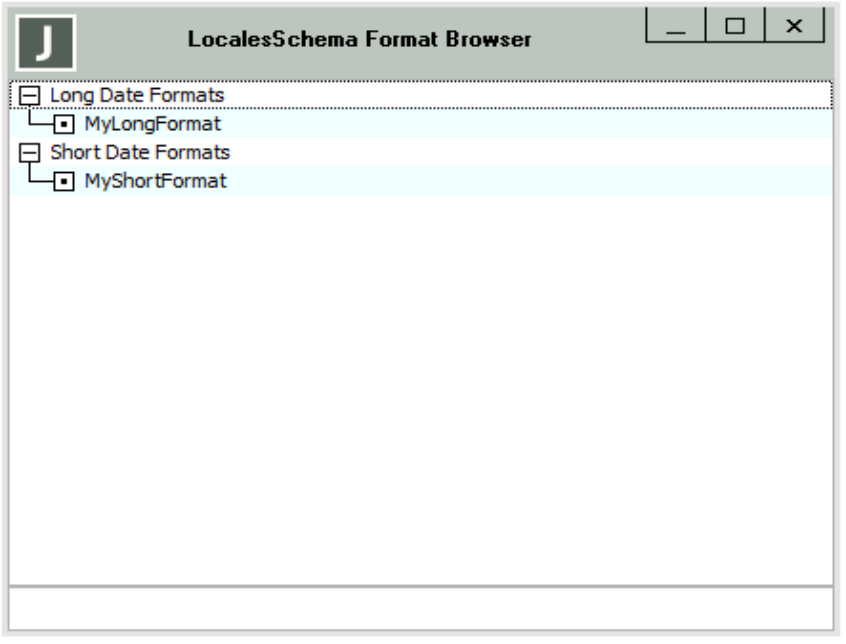
When constructing a picture string for a date format, you will typically use only one of each of the day, month, and year combined with preferred delimiters or other characters, and in your preferred order. However, you can combine delimiters and elements in any manner, including repeating elements; for example, consider the following JadeScript **dateExample** method.

```
dateExample();

vars
    today : Date;
begin
    write today.format("dddd, the dth of MMMM yyyy");
end;
```

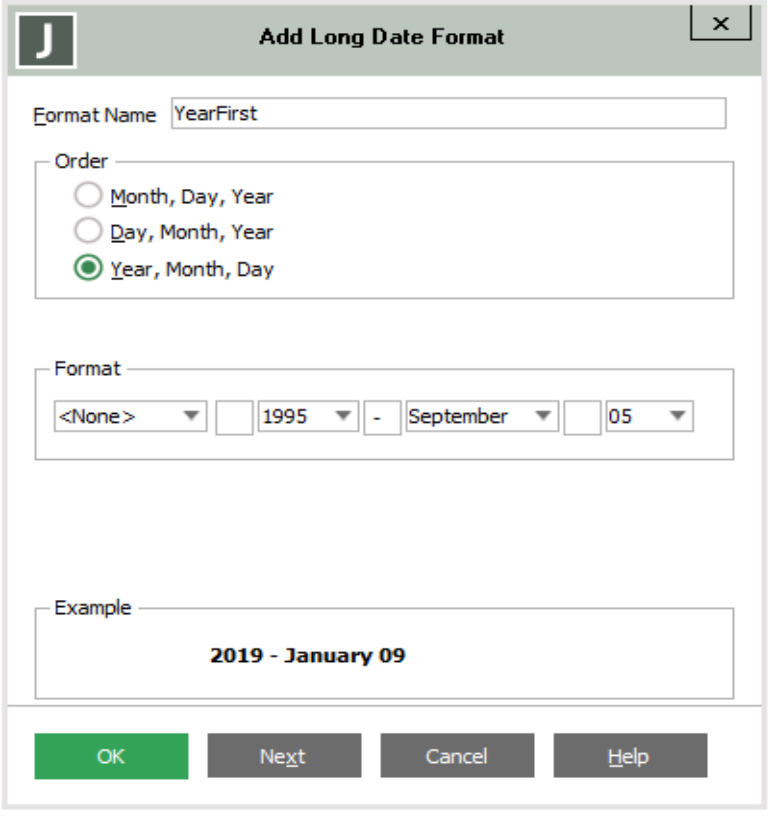
In this method, the day of the month is presented in both full name and single digit, and words are used to form a sentence containing the date. Running the method produces the following output.



As the usage of user-defined date formats increases, it is useful to create and save common formats by using the Format Browser, accessed by selecting the **Formats** command from the **Schema** menu.



The Format Browser provides the ability to create date formats via a form, label them with a descriptive name, and then use them multiple times in a schema by using the **userFormat** method of the **Date** primitive type.

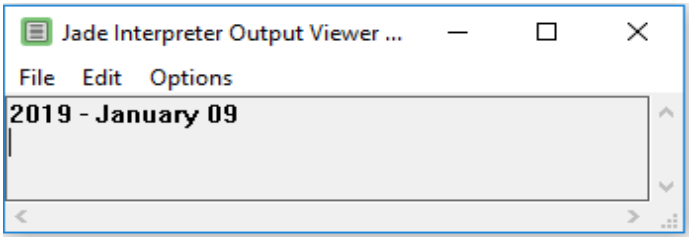


When a date format is created, it can be added to a JADE method by passing it to the **userFormat** method of the **Date** primitive type.

To access a created date format, prefix the format name with a dollar sign (**$**), as shown in the following JadeScript method.

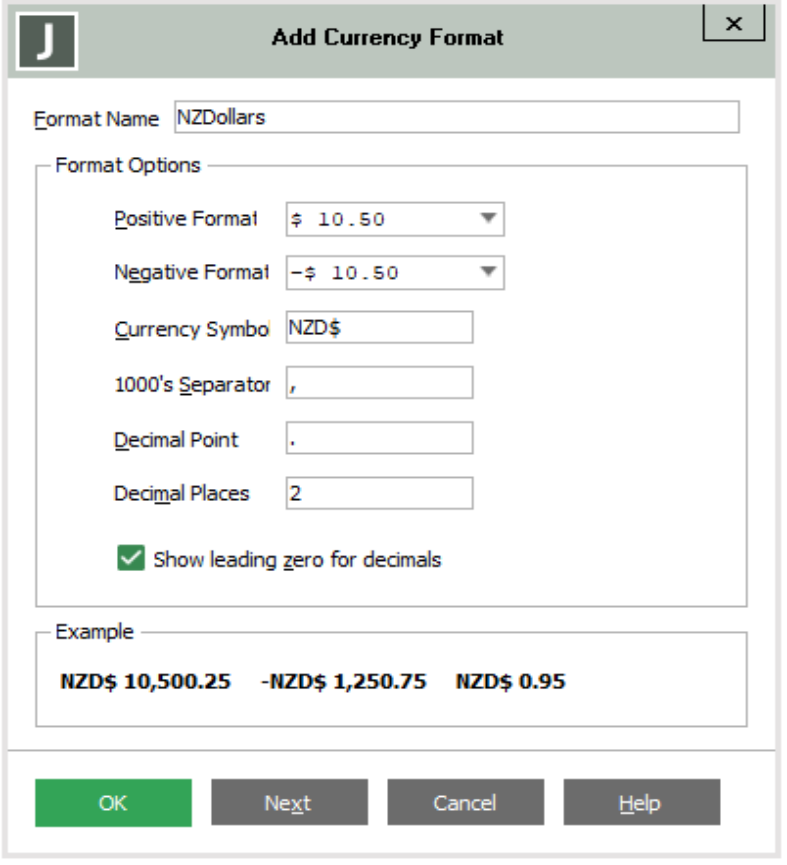
```
dateExample();

vars
    today : Date;
begin
    write today.userFormat($YearFirst);
end;
```

Running the method produces the following output.



Currency Formats

Like date formats, JADE uses the currency format of the operating system by default if no currency format is specified. To specify a custom currency format, you can add a new currency format using the Format Browser and add it to a JADE method using the **userCurrencyFormat** method of the **Decimal** primitive type.

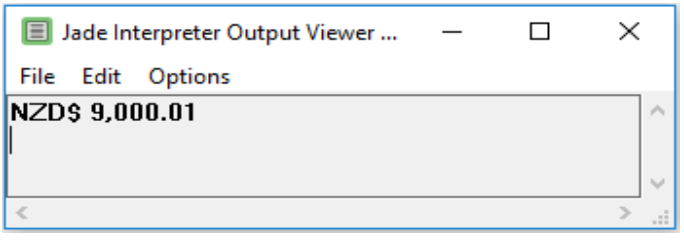


When a currency format is created, it can be added to a JADE method by passing it to the **userCurrencyFormat** method of the **Decimal** primitive type.

To access a created currency format, prefix the format name with a dollar sign (**$**), as shown in the following JadeScript method.

```
currencyExample();

vars
    money : Decimal[12,2];
begin
    money := 9000.01;
    write money.userCurrencyFormat($NZDollars);
end;
```

Running the method produces the following output.



Exercise 6 – Adding a Date Format

In this exercise, you will create several date formats and present the same date in different ways, using the created date formats.

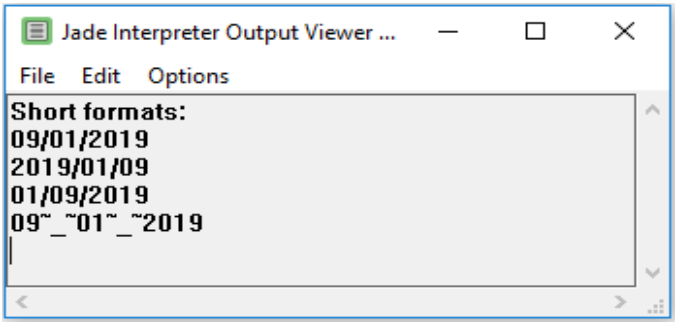
1. With **InternationalSchema** selected in the Schema Browser, open the Format Browser by selecting the **Formats** command from the Schema menu.
2. Add the following short date formats. (You can right-click and then select **Add Short Date Format** from the popup (or context) menu or select the **Add Short Date Format** command from the Formats menu.)

Name	Delimiter	Format
ShortForwards	/	Day, Month, Year
ShortBackwards	/	Year, Month, Day
ShortUS	/	Month, Day, Year
ShortCrazyDelimiter	~_~	Day, Month, Year

3. Add a JadeScript method called **testDateFormats** and code it as follows.

```
testDateFormats();

vars
    today : Date;

begin
    write "Short formats:";
    write today.userFormat($ShortForwards);
    write today.userFormat($ShortBackwards);
    write today.userFormat($ShortUS);
    write today.userFormat($ShortCrazyDelimiter);
end;
```

Running the method produces the following output.



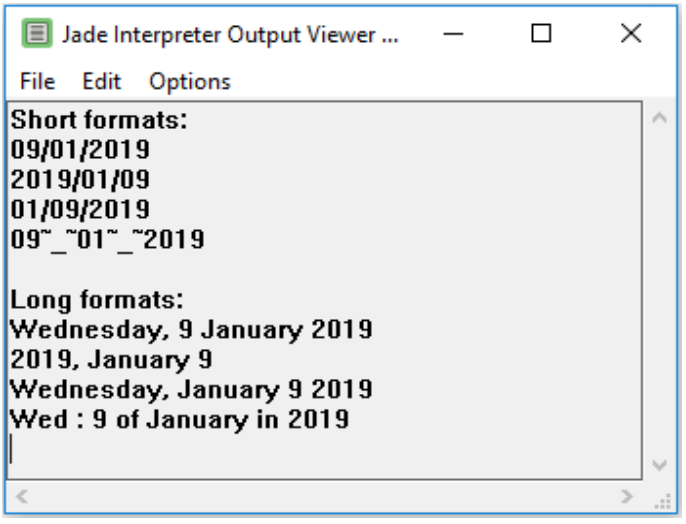
4. Add the following long date formats from the Format Browser.

Name	Format
LongForwards	Format: Sunday , 5 September 1995
LongBackwards	Format: <None> 1995 , September 5
LongUS	Format: Sunday , September 5 1995
LongCrazyDelimiter	Format: Sun : 5 of September in 1995

5. Modify the **testDateFormats** method as follows.

```
testDateFormats();

vars
    today : Date;

begin
    write "Short formats:";
    write today.userFormat($ShortForwards);
    write today.userFormat($ShortBackwards);
    write today.userFormat($ShortUS);
    write today.userFormat($ShortCrazyDelimiter);

    write CrLf & "Long formats:";
    write today.userFormat($LongForwards);
    write today.userFormat($LongBackwards);
    write today.userFormat($LongUS);
    write today.userFormat($LongCrazyDelimiter);
end;
```

Running the method produces the following output.



6. Modify the **testDateFormats** method as follows.

```
vars
    today : Date;

begin
    write "Short formats:";
    write today.userFormat($ShortForwards);
    write today.userFormat($ShortBackwards);
    write today.userFormat($ShortUS);
    write today.userFormat($ShortCrazyDelimiter);

    write CrLf & "Long formats:";
    write today.userFormat($LongForwards);
    write today.userFormat($LongBackwards);
    write today.userFormat($LongUS);
    write today.userFormat($LongCrazyDelimiter);

    write CrLf & "Custom formats:";
    write today.format("d MMM yyy");
    write today.format("M d yyy");

    // One of these will work, the other will do something strange...
    write today.format("The month is MMMM");
    write today.format("Today is dddd");
    // Why is this?
    // Bonus exercise: Fix the non-working one!
end;
```

7. Before you run the **testDateFormats** method, try and work out which of the format methods will have issues.

Exercise 7 – Adding a Currency Format

In this exercise, you will add several currency formats and present the same decimal amount in different ways, using the created currency formats.

1. With **InternationalSchema** selected in the Schema Browser, open the Format Browser by selecting the **Formats** command from the Schema menu.

2. Add the following currency formats. (You can right-click and then select **Add Currency Format** from the popup (or context) menu or select the **Add Currency Format** command from the Formats menu.)

Name	Fields to Change	Values
Accounting	Negative Format	($10.50)
NZDollars	Currency Symbol	NZ$
Pounds	Currency Symbol	£
Yen	Currency Symbol Decimal Places 1000s Separator	¥ 0 *none*

3. Create a JadeScript method called **testCurrencyFormats** and code it as follows.

```
testCurrencyFormats();

vars
    format  : CurrencyFormat;
    formats : CurrencyFormatArray;
    number  : Decimal[12,2];
    numbers : DecimalArray;
begin
    create formats transient;
    formats.add($Accounting);
    formats.add($NZDollars);
    formats.add($Pounds);
    formats.add($Yen);
    create numbers transient;
    numbers.add(9000.01);
    numbers.add(314.15);
    numbers.add(0.02);
    numbers.add(-1000000);
    numbers.add(3.50);

    foreach format in formats do
        write CrLf & format.getName();
        foreach number in numbers do
            write number.userCurrencyFormat(format);
        endforeach;
    endforeach;

epilog
    delete formats;
    delete numbers;
end;
```

4. Running the method produces the following output.