AdaIntl

Librería de internacionalización y localización de programas en Ada95.

AdaIntl – Internacionalización en Ada95

Índice de contenido

AdaIntl – Internacionalización en Ada95

1- ¿Que es AdaIntl ?

AdaIntl es una librería multiplataforma hecha totalmente en Ada95 y liberada bajo licencia LGPL que permite localizar software multilingüe de forma sencilla.

Para traducir un programa sin ninguna librería para ello, es necesario modificar todas las cadenas de texto del código fuente de forma manual, y volver a compilarlas y linkarlas.
Se pueden crear estructuras complejas que devuelvan cadenas de texto en distintos idiomas, pero cualquier traducción nueva o corrección pasa necesariamente por la edición del código fuente, lo que lo dota de muy poca flexibilidad.

AdaIntl permite de una forma extremadamente sencilla crear archivos de texto donde se guarden todas las cadenas de texto que usa el programa. Para traducir la aplicación a otros idiomas, tan solo será necesario editar esos archivos de texto (manualmente, con otras aplicaciones, etc). De esa forma un mismo binario puede estar en tantos idiomas como se desee, y permite cambiar de uno a otro en tiempo de ejecución.

Funciona de forma muy similar a GNU/gettext, el cual si se quiere usar en Ada se debe importar desde C (como hace GTKAda.Intl). AdaIntl evita el tener que instalar gettext o importar y depender de código externo a Ada. AdaIntl es directo, sencillo y fácil de usar.

Es importante resaltar que AdaIntl **NO** traduce el programa, AdaIntl **NO** es un traductor.

Resumen de características:
- Guarda las cadenas de texto en archivos para ser traducidas fácilmente.
- 100% hecho en Ada95.
- Multiplataforma.
- No usa variables de entorno.
- Posibilidad de cambiar de idioma en tiempo de ejecución.
- Guarda el idioma elegido en un archivo para cargar ese idioma en la próxima ejecución.
- 2 tipos de estructura de archivos: en carpetas o en el mismo directorio.
- Identificación de las cadenas de texto mediante el hash “Elf”.
- 6 modos de Debug.
- Posibilidad de adaptar y convertir a archivos *.po.
- Soporte para 175 idiomas.

2- ¿Cómo funciona?

AdaIntl guarda las cadenas de texto del código fuente en archivos de texto (llamados "dominios") para poder ser traducidas y posteriormente leerlas.
Las cadenas de texto que se guardan son las que llevan "-" antes de cada string. Por ejemplo, una cadena que no guardaría sería:

```
Put_Line("Cadena de texto");
```

En cambio sí que se guardaría en el dominio la siguiente cadena:

```
Put_Line(-"Cadena de texto");
```

Dado que las cadenas de texto van a ser traducidas, se calcula un hash para identificarlas en el archivo de localización. Por tanto las cadenas que tengan el mismo hash se consideran iguales.

Al ejecutar el programa, AdaIntl calcula el hash de **"Cadena de texto"** y mira si está en el archivo de localización. Si no es así, lo escribe en él. Si el hash está en el dominio, lee la cadena de texto del fichero (identificada por el hash, ya que la cadena puede variar al ser traducida) y la devuelve para ser visualizada por pantalla.

Para acelerar el proceso, las cadenas de texto de cada dominio están cargadas en memoria y almacenadas en un árbol AA.

Para organizar mejor la traducción, AdaIntl permite usar varios dominios. Por ejemplo, puede haber un dominio "Mensajes_de_error", otro "Avisos", otro "Acerca_de", etc. En cualquier momento se puede pasar de un dominio a otro, ya sea especificando el dominio por defecto, o especificando explícitamente que dominio usar con la función "-".

Además permite elegir el idioma y cambiarlo en cualquier momento, guardar un archivo de configuración con el idioma elegido por el usuario (y así usar ese mismo idioma en la próxima ejecución), obtener todos los idiomas en los que está disponible la aplicación, etc.

3- Descripción de los tipos y funciones de AdaIntl

3.1- Tipos

3.1.1 – Internationalization_Type

Este tipo tan solo se usa para poder inicializar AdaIntl durante la etapa de declaración del programa. No vuelve a usarse más adelante (excepto para volver a inicializarlo si hace falta).
La función encargada de inicializar AdaIntl es "**Initialize_AdaIntl**" y se explica en el apartado 3.2.1.

¿Porque puede ser necesario inicializar AdaIntl en la etapa de declaración?
Por ejemplo si nuestro programa tiene estas lineas:

```
        Mensaje: String:= -"Hola";
begin
        Initialize_AdaIntl(...);
```

En este caso primero se ejecutaría "-" con Hola, pero al no estar AdaIntl inicializado, daría un error. En cambio podemos ejecutar antes "**Initialize_AdaIntl**" y de esta forma asegurarnos una ejecución correcta:

```
        IT: Internationalization_Type:=Initialize_AdaIntl(...);
        Mensaje: String:= -"Hola";
```

Aparte de permitir la ejecución de "**Initialize_AdaIntl**", el tipo "*Internationalization_Type*" no tiene otro propósito.

Si no se quiere inicializar AdaIntl durante la etapa de declaración, se puede usar el procedimiento "**Initialize_AdaIntl**", el cual es exactamente que la función del mismo nombre, y de esta forma no hay que declarar ni usar ninguna variable de tipo "*Internationalization_Type*".

3.1.2 – Debug_Level_Type

AdaIntl dispone de 6 modos de uso y debug:

- **Deactivated**
- **No_Debug**
- **Only_Errors_Stop**
- **Only_Errors_No_Stop**
- **Total_Stop**
- **Total_No_Stop**

- El modo "*Deactivated*" desactiva el sistema de localización. No se hace nada, y la función "-" devuelve la misma cadena de entrada.

- El modo "*No_Debug*" hace funcionar la librería de forma normal. Si hay algún fallo, no imprime ningún valor ni da ningún aviso, sino que "-" devuelve la misma cadena de entrada. Este modo "*No_Debug*" es el modo por defecto y es el que debería ser usado cuando el programa esté listo en la fase de explotación.

- "*Only_Errors*": Este modo tan solo imprime mensajes de error cuando hay fallos (archivos de traducción incorrectos, etc). Hay dos variantes, "*Only_Errors_No_Stop*" y "*Only_Errors_Stop*". El primero (No_Stop) no detiene la ejecución del programa, ante un error "-" devuelve la cadena de entrada. El modo Stop lanza una excepción que detiene la ejecución del programa. Estos modos se pueden usar durante la desarrollo y testeo del programa, aunque también se podría usar durante la fase de explotación.

- Por último, "Total" imprime todo tipo de mensajes diciendo que hace y deja de hacer AdaIntl. También dispone de dos submodos "Stop" y "No_Stop" que continua la ejecución o la detiene con una excepción. Este modo se usa para comprobar el buen funcionamiento de AdaIntl.

3.1.3 – Language_Type

AdaIntl tiene soporte para 175 idiomas, contenidos en el paquete **d_idiomas**.

Language_Type es un tipo creado a partir de *d_idiomas.T_Language* para ofrecer visibilidad cara al programador. Cualquier cambio que se desee realizar en los idiomas se han de hacer en ese paquete **d_idiomas**.

Los idiomas son los enumerados en el ISO 639-1.

Sin embargo, cuenta con 2 diferencias:

- Algunos idiomas tienen 3 letras, como Isl y Ori, porque su código ISO es una palabra reservada en Ada95 (en estos casos, **IS** y **OR**).
- Se añade un idioma "comodín". Es el idioma **NULO** (nul – no se usa **null** por ser palabra reservada en Ada95). Este debe ser el *ULTIMO* en la lista enumerada. Si se añadiesen otros idiomas, se deben colocar antes que nul: deberá seguir siendo el último para el correcto funcionamiento de **AdaIntl**.

3.1.4 – Available_Languages_Array

Esta estructura permite saber que idiomas hay disponibles.

Es un vector de *booleans*, que dado un idioma "*Language_Type*" (vease 3.1.3) devuelve "*true*" si existe un fichero de traducción para ese idioma, y "*false*" si no existe.

Se usa con la función "*Get_Available_Languages*". Para más información y códigos de ejemplos, véase el punto 3.2.11.

3.2- Funciones

3.2.1 – Initialize_AdaIntl (función)

```
function Initialize_AdaIntl (
    Language                : Language_Type;
    Default_Domain          : String         := "Language";
    Debug_Mode              : Debug_Level_Type := No_Debug;
    Directory               : String         := "";
    Load_Configuration_File : String         := ""         )
 return Internationalization_Type;
```

Lo primero que hay que hacer es inicializar AdaIntl. Para ello antes de nada se debe ejecutar la instrucción "**Initialize_AdaIntl**", con una serie de argumentos. Es *obligatorio* ejecutar "**Initialize_AdaIntl**" antes que cualquier otra función o procedimiento de AdaIntl. "**Initialize_AdaIntl**" es una función y no un procedimiento para poder ser ejecutada antes que cualquier cadena de texto (tal como se explica en el punto 3.1.1).

Los argumentos de **Initialize_AdaIntl** son:

- **Idioma** (**Language**):
  Es necesario especificar un idioma, que será el usado a partir de ese momento.

- **Dominio por defecto** (**Default Domain**):
  También se debe especificar en que archivo se guardarán las cadenas. Puede haber varios dominios, pero solo uno será el dominio "por defecto". Si no se especifica nada, el archivo será "*Language*".

- **Debug**:
  AdaIntl permite varios modos de uso y debug, tal como se especifica en el punto 3.1.2.

- **Directorio** (**Directory**):
  Se puede especificar que estructura de fichero usarán los dominios. Hay 2 tipos disponibles: todos los ficheros de localización en el mismo directorio que el programa, u organizados en carpetas por idiomas.

  Si "*Directorio*" es "" (cadena nula), los ficheros se organizan de la primera manera (todos en el mismo directorio y diferenciados por extensión).
  Por ejemplo, si tenemos el programa "Aplicacion" y el dominio "Language" en inglés y castellano, el directorio del programa tendrá los siguientes archivos:

  - *Aplicación* (binario del programa)
  - *Language.ES* (dominio en castellano)
  - *Language.EN* (dominio en inglés).

Este modo es útil cuando la aplicación es pequeña y tiene pocos dominios.

Por otro lado, se puede organizar por directorios. Para ello "*Directorio*" ha de tomar un valor distinto a la cadena nula y será el directorio donde se guardarán los dominios. Es **muy importante** que el último carácter sea el de separación de directorios (/). Por ejemplo, "language/" con el caso anterior crearía los siguientes ficheros:

- *Aplicación* (binario del programa)
- *Language* (directorio)
  - *ES* (directorio de dominios en castellano)
    - *Language* (dominio en castellano)
  - *EN* (directorio de dominios en inglés)
    - *Language* (dominio en inglés).

Es importante notar que los ficheros en este caso NO tienen extensión (en el modo anterior tenían extensión para diferenciar el idioma, como ahora se diferencian por el directorio, pueden tener la extensión que el usuario quiera).
Este modo es útil cuando la aplicación es grande y tiene muchos dominios, ya que organiza mejor los ficheros de localización. Sin embargo presenta un **problema**: Ada95 **no** permite crear directorios, por lo que antes de ejecutar la aplicación se deberán crear manualmente. De lo contrario fallará y dependiendo del nivel de Debug detendrá la ejecución y "-" irá devolviendo las cadenas de entrada sin localizar el programa. Ada2005 sí permite manipulación de directorios.

- **Ruta del fichero de configuración** (**Load_Configuration_File**):
  Por último, se puede cargar el idioma de un fichero con la ruta especificada.
  Si el fichero ya existe, se leerá y se usará el idioma indicado en ese fichero y se ignorará el *Idioma* especificado en el primer parámetro. Si el fichero es incorrecto, y dependiendo del modo de debug especificado anteriormente, abortará la ejecución o usará el *Idioma* especificado en el primer parámetro.
  Si el fichero no existe, usará el *Idioma* especificado en el primer parámetro y creará el fichero de configuración con ese idioma.

3.2.2 – Initialize_AdaIntl (procedimiento)

```
procedure Initialize_AdaIntl (
    Language                : Language_Type;
    Default_Domain          : String          := "Language";
    Debug_Mode              : Debug_Level_Type := No_Debug;
    Directory               : String          := "";
    Load_Configuration_File : String          := ""          );
```

El procedimiento "*Initialize_AdaIntl*" tiene el mismo efecto que la función "*Initialize_AdaIntl*". La diferencia, tal como se explica en el punto 3.1.1, es que la función "*Initialize_AdaIntl*" sirve para iniciar AdaIntl en la fase de declaración. Aparte de eso, los parámetros y su uso es similar. Para más información, véase el punto 3.2.1.

3.2.3– Set_Default_Domain

```
procedure Set_Default_Domain (
    Domain : String );
```

Una vez ejecutado "*Initialize_AdaIntl*", es posible cambiar el dominio por defecto en cualquier momento ejecutando este procedimiento. Tan solo se ha de indicar cual es el nombre del archivo de traducción (dominio), y a partir de ese momento se usará ese dominio cuando la instrucción "-" (vease 3.2.9) no especifique un dominio.

3.2.4– Set_Debug_Mode

```
procedure Set_Debug_Mode (
    Debug_Mode : Debug_Level_Type );
```

Una vez ejecutado "*Initialize_AdaIntl*", es posible cambiar el nivel de Debug en AdaIntl . A partir de ese momento se usará el nivel de Debug especificado por el parámetro "*Debug_Mode*".
Los distintos niveles se pueden ver en el punto 3.1.2.

3.2.5– Set_Language

```
procedure Set_Language (
    Language : Language_Type );
```

Una vez ejecutado "*Initialize_AdaIntl*", es posible cambiar el idioma por defecto en tiempo de ejecución con esta instrucción. El parámetro "*Language*" puede tener cualquier valor especificado en el punto 3.1.3.
Ejecutando "*Set_Language*", **no** se actualizan las frases cargadas en memoria, sino que se actualizarán a medida que se vayan usando los dominios (al acceder a un dominio se compara el lenguaje de los dominios cargados en memoria con el lenguaje por defecto, si no coinciden, se actualiza el dominio en cuestión).

3.2.6– Get_Default_Domain

```
function Get_Default_Domain return String;
```

Esta función permite obtener el dominio por defecto que se está usando actualmente.

3.2.7– Get_Debug_Mode

```
function Get_Debug_Mode return Debug_Level_Type;
```

Esta función permite obtener el nivel de Debug que se está usando actualmente (vease el punto 3.1.2).

3.2.8– Get_Language

```
function Get_Language return Language_Type;
```

Esta función permite obtener el idioma por defecto que se está usando actualmente. Especialmente útil debido a que AdaIntl puede cargar cualquier idioma de un fichero de configuración (véase 3.2.1, último parámetro).

3.2.9– “-” (un parámetro)

```
function "-" (
    Right : String )
 return String;
```

Una de las funciones más importantes y que más se usarán de AdaIntl. Es el equivalente a las funciones de GNU/gettext “*gettext()*” y “*_()*”.

Esta función, al ser ejecutada, obtiene la cadena a traducir de su único parámetro.
A continuación obtiene su traducción según el idioma por defecto que esté activo en ese momento y devuelve la cadena traducida.
El dominio usado es el dominio por defecto.

Para obtener la cadena traducida, mira en memoria en el dominio por defecto si está almacenada. Si no es así, abre el fichero de traducción (dominio) y carga las frases del dominio.
Si aún así sigue sin estar presenten, entonces *añade* la frase sin traducir al dominio y la devuelve tal cual.

Esto significa que el archivo de traducción se crea *en tiempo de ejecución*. A medida que se vaya ejecutando el programa, y se ejecute “-” con las cadenas de texto, estas se irán guardando en los ficheros de traducción (si estos no existen). Si una linea no se ejecuta, la cadena de texto contenida **no estará** presente en los archivos de traducción.

Se desprende que por tanto la única forma (actualmente) de crear los archivos de traducción por primera vez es ejecutando la aplicación y asegurándose de que se ejecuten todos los “-”.

Si no se ejecutan todos (porque la aplicación falle, o se añadan nuevas cadenas), las nuevas aparecerán al final del archivo de traducción. Por tanto si se dispone de una traducción parcial, tan solo habría que añadir las nuevas lineas al final de cada archivo traducido (y traducir las nuevas cadenas).

Se pueden realizar programas externos, que al igual que xgettext, lean el código fuente y extraigan automáticamente las cadenas a traducir, creando los ficheros adecuados, y de esa forma no haga falta “ejecutar” completamente la aplicación.
Sin embargo, por el momento no hay ninguna aplicación que lo haga.

Ejemplo de uso de “-”:

```
Put_Line(-"Probando, probando");
```

3.2.10- “-” (dos parámetros)

```
function "-" (
      Left  : String;
      Right : String  )
  return String;
```

Esta función es equivalente a “-” con un solo parámetro, pero con la particularidad de que se ha de especificar el dominio a usar. El parámetro “*left*” es el dominio a usar, y “*right*” es la cadena de texto.
En lugar de usar el dominio por defecto, usará el especificado en “*left*”.

Ejemplo de uso de “-” especificando el dominio:
Queremos mostrar por pantalla “Probando, Probando”, y guardar la cadena en el dominio “Lista_de_saludos”.

```
Put_Line("Lista_de_saludos"-"Probando, probando");
```

3.2.11- Get_Available_Languages

```
function Get_Available_Languages return Availabe_Languages_Array;
```

Esta función permite conocer en que idiomas está disponible la aplicación. Para ello mira si existe el fichero del dominio por defecto para cada idioma, y devuelve una estructura "*Available_Languages_Array*" (3.1.4), donde "*true*" significa que existe el fichero del dominio por defecto y "*false*" que no existe.
El idioma "*nul*" **siempre** devuelve "*false*".

Ejemplo de uso de "Get_Available_Languages":
Queremos mostrar por pantalla los idiomas disponibles en los que está nuestra aplicación:

```
   Idiomas_Disponibles : Availabe_Languages_Array;
begin
   Initialize_AdaIntl (Es);
   Idiomas_Disponibles:=Get_Available_Languages;
   Put_Line("Los idiomas disponibles son:");
   for Iterador in Language_Type'range loop
      if Idiomas_Disponibles(Iterador) then
         Put_Line(Language_Type'Image(Iterador));
      end if;
   end loop;
```

3.2.12- Clean_AdaIntl

```
procedure Clean_Adaintl;
```

La instrucción Clean_AdaIntl se ocupa de borrar de la memoria las frases, destruyendo los árboles y su contenido.
Se **debe** ejecutar esta instrucción antes de finalizar el programa o bien si se desea "reiniciar" AdaIntl en tiempo de ejecución (justo antes de volver a hacer "*Initialize_AdaIntl*").