
ZanyBlue Documentation

Release 1.3.0b

Michael Rohan <mrohan@zanyblue.com>

Aug 06, 2016

CONTENTS

1	Globalization Support	3
2	Parameter Handling	5
3	Testing Application	7
4	License	9
5	ZanyBlue Documentation	11
5.1	ZanyBlue License	11
5.2	Downloads	11
5.3	Introduction	11
5.4	Releases	13
5.5	The Root Package	15
5.6	The Text Package	15
5.7	Command Line Utilities	50
5.8	Additional Documentation	52
5.9	Contributions	52
5.10	Notices	52
5.11	References	73
6	Indices and tables	75
	Bibliography	77

This documentation covers version 1.3 of the ZanyBlue library and applications. For download information see [Downloads](#). If you encounter bugs in this software, please file reports in the [Source Forge Bug Tracker](#). You can also visit the [Source Forge ZanyBlue Project Page](#).

GLOBALIZATION SUPPORT

The initial functionality covers globalization support for Ada. Globalization, (normally abbreviated to simply `g11n`) covers support within the source code for multiple languages (this is termed internationalization or `i18n`) and the ability to supply translations for strings used in an application (localization or `l10n`). Globalization is supported by the ZanyBlue text library by providing routines to access and format messages in Ada sources (the `i18n` aspect of `g11n`) and localization via the externalization of messages strings to properties files. For example, the following properties file defines a simple message referred to in the source code via the key `Hello`:

```
Hello=Hello World, I am {0}
```

An Ada application code could print this message using the accessor routine

```
Name : constant String := "Michael";  
...  
Print_Hello (+Name);
```

See *The Text Package* documentation and *The `zbmcompile` Utility* application.

PARAMETER HANDLING

To support command line handling and user defined parameters and scopes in the `zbttest` application, the `Parameters` packages are available, e.g.,

```
Parameters : Parameter_Set_Type;  
...  
Parameters.Set_Boolean ("verbose", True);  
if Parameters.Get_Boolean ("verbose") then  
    Print_Debug_Banner;  
end if;
```


TESTING APPLICATION

The stability of the code based is improved via a large set of tests. These tests are split between AUnit based unit-tests and a set of tests designed to exercise the ZanyBlue utilities, more black-box style testing. These tests use the `zbttest` utility. There is no formal documentation on this at this time other than the built-in help and the example test scripts in the `tests` directory.

LICENSE

ZanyBlue is distributed under the terms of a BSD-style license. See [ZanyBlue License](#) for details.

ZANYBLUE DOCUMENTATION

ZanyBlue License

The ZanyBlue library and utilities are released under the BSD license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of ZanyBlue nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the copyright holders and contributors “as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the copyright holder or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

Downloads

The current release is 1.3. The package is available as a Unix gzipped tar file (Windows users can use 7-Zip to extract):

- Latest Unix gzipped tar ball [download](#).

You can also browse the [downloads area](#) on [SourceForge](#) or checkout the latest code checked into Subversion:

```
$ svn co https://svn.code.sf.net/p/zanyblue/code/trunk zanyblue
```

Introduction

This manual describes the use of the ZanyBlue libraries, an Ada framework for finite element analysis with supporting libraries which might be of general use in other projects. The initial implementation contains the `Text` package (and supporting packages and utilities) supplying functionality mirroring the Java `MessageFormat` package. The `Text`

functionality allows the application messages to be externalized in `.properties` files and localized into additional languages. As a major side effect the formatting of arguments within messages is needed and implemented.

This, naturally, led to a need for a testing framework. Ahven is used for unit testing of the libraries, however, testing of the command line utilities proved more difficult. While DejaGNU was available, it seemed a little more involved than was needed for this project. A new testing application was written, `zctest`, which is a hierarchical testing framework supporting definition scopes.

Building the Library and Applications

Building the ZanyBlue library and applications does not require a configure step (at this time). The build requirements are similar on Unix (Linux) and Windows (the Makefiles use the environment variable `OS` to determine whether the build is Linux (the variable is not defined) or Windows (where it is defined with the value `Windows_NT`).

Requirements

The build requires

- A recent version of GNAT (e.g., GPL 2010, 2011) from AdaCore.
- GNU make.
- The XMLAda package needs to be installed to build the ZanyBlue testing application, `zctest`.
- The unit tests require the Ahven package

Building

Once the required environment is in place, building is simply a matter of issuing `make` in the `src` directory, e.g.,:

```
$ make -C src
```

This produces the library in the `lib/zanyblue` directory, and the two executables `zcmcompile`, the message compiler, `zctest` the ZanyBlue testing application and `zbinf` a utility to query built-in library configuration. The ZanyBlue package spec files and generic bodies are copied to the `include/zanyblue` directory.

For example, to examine the built-in locale data, e.g., to see the date, time, numeric, etc data for Swedish, use the `zbinf` command:

```
$ zbinf --dump-locale sv
```

Using with GNAT

A ZanyBlue GNAT `gprbuild` project file is available in the directory `lib/gnat` directory. Adding this directory to your `ADA_PROJECT_PATH` should allow the use of the ZanyBlue library with `gprbuild` projects. See the text examples and the GPS build instructions.

Testing

Testing is via the `check` target. From the `src` directory this will run both the unit and system tests. If experimenting, running these tests independently is probably more useful (the combined testing is normally only run from under Jenkins where the summary XML files are loaded as the test results for the build):


```
$ make -C src/test/unittest check
$ make -C src/test/system check
```

Examples

The examples include a Gtk example. If Gtk is not installed, this example will fail to build (but the other should be OK). The `dumplocale` from previous releases has been replaced by the `zbinfo` utility.

Windows Issues

The installation of the Windows [GNU Win32 version of make](#), does not update the Path environment. This should be done manually after installing via the Control Panel -> System -> Advanced -> Environment Variables -> Path and adding the path `C:\Program Files\GnuWin32\bin`.

The majority of the example application are text based. This is problematic on Windows systems as the standard Windows command console does not understand Unicode output. This makes the Gtk example application the only fully functional example application on Windows systems. To build this application, the GtkAda package should be installed, again from AdaCore and, as for AUnit, the `gprbuild` project path environment variable will need to be set if GtkAda is not installed in the default location.

Releases

The ZanyBlue releases are distributed when functionality has been completed. The following section detail the contents of the releases.

Version 1.3.0b, Aug 6th, 2016

1. Added a new utility `zbinfo` to query built-in data. This was released as an example previously (the `dumplocale` example which has been dropped).
2. Added encoding support to convert `Wide_String` values to `String` based on an encoding schema, e.g., UTF-8, ISO8859-2, CP932, SHIFT_JIS, etc. To fully use this functionality, narrow accessors should be used which, when printing, use `Stream_IO` to avoid interaction between the `Text_IO` and encoded values. The list of supported encodings is available via `zbinfo --list-encodings`.
3. The default locale is now `en_US.UTF-8` if no other locale can be determined from the environment.
4. Updated the documentation (and website) to use the Sphinx documentation system.
5. Updated and expanded the documentation. Additional documentation is, however, needed. Switch to `gnatdoc` from `gnathtml` to generate the source code based documentation.
6. Restricted the usage of the `-gnatW8` compilation option to just the source files containing UTF-8 encoded strings: the message pool file generated by the `zbmcompile` utility.
7. Added option to the `zbmcompile` utility to generate ASCII only source files (`-A` option).
8. Added option to the `zbmcompile` utility to define handling of non-Ada message keys when generating accessors (the `-X` option).
9. Updated the build to use `gprbuild` instead of `gnatmake`.
10. Updated the build to use `-gnat2012` in all `gpr` files.

11. Switched from AUnit to Ahven for unit testing.
12. Minor source code changes based on stricter checks with GNAT 2016.

Version 1.2.1b, Sep 13th, 2015

1. Minor patch update to handle stricter checks with the latest GNAT release

Version 1.2.0b, Mar 10th, 2014

1. Updates for building with GNAT 2013
2. Moved usage of Generic packages to library level (stricted accessibility checks with GNAT2013)
3. Dropped definition of “ld run path” definition (created issues on MacOS)
4. Updated CLDR data to v24 Release (see cldr.unicode.org)
5. Allow localization for +/- characters to be multi-character strings
6. Improved errors messages for invalid zbmcompile command line arguments
7. Implemented message filtering for all Print routines
8. Added directory tree level initialation files for zbtest
9. Bugfixes, e.g., handling OS LANG values with dashes in the encoding
10. Some documentation updates
11. Added a patch to ZanyBlue-ize GNAT GPS 5.2.1 (released with GNAT 2013)

Version 1.1.0b, Jul 2nd, 2012

1. Updates for building with GNAT 2012 (major driver for this release)
2. Added a patch to ZanyBlue-ize GNAT GPS 5.1.1 (released with GNAT 2012) The resultant executable is identical to the standard gps but with support for pseudo translation (no real attempt is made to supply localized properties files for gps)

Version 1.0.0b, Apr 30th, 2012

Major changes in this release are:

1. Change of licensing from GPLv2 to simple BSD
2. Message accessors to make using messages safer (the Ada compiler can check for argument types and number)
3. A simple regression testing application for command line utilities
4. A parameter storage package (internally used for command line and for parameter storage during regression tests)

Version 0.1.0b, Nov 26th, 2010

Initial release licensed under GPL 2.0.

The Root Package

The ZanyBlue root package, `ZanyBlue` contains definitions about the library, primarily version information. The version numbers associated with the library are available via the functions

- **Version_Major**, the major version number for the `ZanyBlue` release.
- **Version_Minor**, the minor version number for the `ZanyBlue` release.
- **Version_Patch**, the patch number for the `ZanyBlue` release.

Each release also has a release status given by the function:

```
function Version_Status_Type return Version_Status_Type;
```

The return value is an enumeration type: Alpha, Beta, Production.

To further refine the release identification, a source code revision value (a wide string) is returned by the function `Revision`. The source code system is currently Subversion and this string has the format of the letter `r` followed by the Subversion revision number, e.g., `r2500`.

Finally, the copyright year for the bundle is given by the function:

```
function Copyright_Year return Positive;
```

This is current year at the time of the build.

The Text Package

The intent of the `Text` package is to allow the separate definition of the text messages displayed by an application into a `.properties` file. Messages are reference in the source code by a key value (a simple string or accessor function returning the text associated with a particular key) within a group of related messages – a facility. The message strings include Java style message argument place holders: argument number (zero-based) along with optional type specific formatting information enclosed in chain brackets. Allowing formatting of message arguments in the context of a localized message.

The divorce of the message text from the source code allows the easy localization of application messages. The source language `.properties` file is delivered to translation vendors who then return localized `.properties` files. E.g., for an application `myapp`, with application messages defined in the file:

```
myapp.properties
```

the localized French and Japanese files would be:

```
myapp_fr.properties
myapp_ja.properties
```

The `Text` package supports message text searching among a set of localized `.properties` files with fallback to available messages. For example, if the above application, `myapp`, is run in a Canadian French environment (`ZB_LANG=fr_CA` on Unix), localized display of each message is implemented by trying to locate the message in the files, in order:

```
myapp_fr_CA.properties
myapp_fr.properties
myapp.properties
```

If no French localized message is available, i.e., the properties file which does not include a language code is used, the base properties file. Frequently this is English (the final file shown above).

Contents:

Messages and Arguments

The localization of a message cannot, in general, occur using just the text “snippets” surrounding values generated by an application, e.g., the externally visible message:

```
There are 2 moons orbiting "Mars".
```

could be divided into the “snippets”

- There are
- moons orbiting "
- ".

concatenated together at runtime along with formatted values for the number of moons and the planet name. If delivered to a translation vendor, the “snippets” would need to localize independently. This is a rather difficult task to do for general text “snippets” as the order of the “snippets” is defined by the application and cannot be changed by localization vendors. Also, the “snippets” themselves are generally too short to give translators sufficient context. As a general rule, translation should be performed on complete sentences.

A complete sentence is possible if place-holders are used for message arguments. Using Java style arguments, the above message would be defined as:

```
There are {0} moons orbiting "{1}".
```

The application message arguments are referenced within chain brackets. In the example here, argument 0 would be the integer 2 and argument 1 would be the string Mars.

This message is then stored in a `.properties` file, external to the application and given a key name, e.g., to use the key string `moons` to refer to the message in the application code, the `.properties` file definition would be:

```
moons=There are {0} moons orbiting "{1}".
```

This sentence can be localized independently of the application and, since the English string is also defined externally to the application, this text can also be changed. For example, if a technical writer review of the English used in an application decided, for consistency reasons, to rephrase the sentence, just the English properties file need be adjusted, e.g.,:

```
moons=The planet "{1}" has {0} moons.
```

This model of searching `.properties` files is a conceptual model in the context of the ZanyBlue Text library. The ZanyBlue implementation compiles the set of properties files for a facility into Ada code and localized message resolution for a particular key occurs against run-time data structures rather than accessing files. This makes for an efficient implementation where the major portion of the data processing occurs at application build time. See details on the `zbmcompile` (*The `zbmcompile` Utility*), the ZanyBlue message compiler, later.

Example Using Low Level Functions

As an introduction to the `Text` package, a simple example, which continues the example based on the number of moons orbiting the various planets, is developed here.

Note, the preferred way to access messages is using accessor packages described in the next section. The example here uses direct calls using strings to identify facilities and key to demonstrate the concepts involved.

The task is to ask the user for a planet name and print the number of currently known moons for the planet. The source for this simple example is available in the directory `examples/text/moons` and is listed here:

```
with Ada.Wide_Text_IO;
with Moons_Messages;
with ZanyBlue.Text.Formatting;

procedure Moons is

    type Planet_Names is (Mercury, Venus, Earth, Mars,
                          Jupiter, Saturn, Uranus, Neptune);
    package Planet_Name_IO is
        new Ada.Wide_Text_IO Enumeration_IO (Planet_Names);

    use Ada.Wide_Text_IO;
    use Planet_Name_Formatting;
    use ZanyBlue.Text.Formatting;

    Moons : array (Planet_Names) of Natural := (
        Earth => 1, Mars => 2, Jupiter => 63,
        Saturn => 62, Uranus => 27, Neptune => 13,
        others => 0);
    Planet : Planet_Names;

begin
    loop
        Print ("moons", "0001");
        Planet_Name_IO.Get (Planet);
        if Moons (Planet) /= 1 then
            Print_Line ("moons", "0002", +Moons (Planet),
                      +Planet_Names'Image (Planet));

        else
            Print_Line ("moons", "0003", +Planet_Names'Image (Planet));
        end if;
    end loop;
exception
when End_Error | Data_Error =>
    New_Line;
    Print_Line ("moons", "0004");
end Moons;
```

The example source code uses the `Generic_Enumerations` ZanyBlue package for planet name arguments. Generic ZanyBlue packages should be instantiated at the library level.

Here the messages for the application are referred to by message id or key, i.e., numeric strings, e.g., 0001 for the facility `moons`.

The text associated with these message keys are externalized to a `.properties` file. For this example, the root properties file, `moons.properties`, containing English, is:

```
0001=Please enter a planet:
0002=There are {0} known moons orbiting "{1}".
0003=There is 1 known moon orbiting "{0}".
0004=OK, goodbye.
```

A German translation of this properties, `moons_de.properties`, file would be (via Google Translate):

```
0001=Bitte geben Sie einen Planeten:
0002=Es gibt {0} bekannte Monde umkreisen "{1}".
0003=Es gibt 1 bekannt Mond umkreisen "{0}".
0004=OK, auf Wiedersehen.
```

French and Spanish Google translated version of this file in the examples directory.

The properties and application are tied together by “compiling” the properties files into an Ada package and simply **with**ing the compiled package in the application (normally in the unit containing the main procedure). In this example, the generated package referenced in the source above is `Moons_Messages` which is created using the `zbmcompile` ZanyBlue message compiler utility:

```
$ zbmcompile -i -v Moons_Messages moons
This is ZBMCompile, Version 1.3.0 BETA (r3059) on 8/6/16 at 7:44 PM
Copyright (c) 2009-2016, Michael Rohan. All rights reserved
Loaded 20 messages for the facility "moons" (4 locales)
Performing consistency checks for the facility "moons"
Loaded 1 facilities, 5 keys, 4 locales and 20 messages
Loaded total 677 characters, stored 677 unique characters, 0% saving
Retained the spec "Moons_Messages" in the file "./moons_messages.ads"
Wrote the body "Moons_Messages" to the file "./moons_messages.adb"
ZBMCompile completed on 8/6/16 at 7:44 PM, elapsed time 0:00:00.026
```

This generates an Ada specification containing the name of the facility, `moons` in this case, and the definition of a single initialization routine. The body file contains the message text data, the initialization routine which adds the message data to a shared message pool. Since the `-i` command line option was used, the generated package body includes a call to the initialization routine allowing the message data to be included in the application via a simple **with** of the package, i.e., no explicit call to the initialization routine is needed.

The execution of the generated application will now display messages selected for the run-time locale, e.g., in an English locale:

```
$ make run
.././././bin/x_moons
This is MOONS, Version 1.3.0 - BETA
Please enter a planet: mars
There are 2 known moons orbiting "MARS".
Please enter a planet: earth
There is 1 known moon orbiting "EARTH".
Please enter a planet: mercury
There are 0 known moons orbiting "MERCURY".
Please enter a planet: ^D
OK, goodbye.
```

And running the application selecting a German locale displays the German messages:

```
$ make run_de
.././././bin/x_moons de
Dies ist MOONS, Version 1.3.0 - BETA
Bitte geben Sie einen Planeten: mars
Es gibt 2 bekannte Monde umkreisen "MARS".
Bitte geben Sie einen Planeten: earth
Es gibt 1 bekannt Mond umkreisen "EARTH".
Bitte geben Sie einen Planeten: mercury
Es gibt 0 bekannte Monde umkreisen "MERCURY".
Bitte geben Sie einen Planeten:
OK, auf Wiedersehen.
```

The locale is selected via a command line argument here, however, the environment variables can be used instead, see section *Changing Default Locale* for more details on selecting locales.

This example application did not attempt to localize the planet names. A more involved example would add another facility for the planet names with message keys matching the `Image` of the enumeration values.

Example Using Accessor Functions

The style of message generation using a facility string, key string and message arguments opens an application up to the risk of a mis-match between the expected arguments for a message and the supplied arguments (possibly causing the ZanyBlue exceptions `No_Such_Argument_Error` or `No_Such_Key_Error` to be raised). It is also possible to silently, in the sense the compiler does not fail to compile, introduce errors by accidentally mis-spelling a facility or key value (again possibly causing the exceptions `No_Such_Facility_Error` or `No_Such_Key_Error` to be raised).

For an Ada application, where the compile time detection of errors is an expectation. This situation is not ideal. To overcome this, the ZanyBlue message compiler utility can generate accessor packages for the compiled facilities. These packages define functions and procedures for each key in the facility with a list of arguments matching the expected number of arguments. This allows the Ada compiler to perform checks on the references to messages of ZanyBlue-used applications.

The `zbmcompile` utility generates three major classes of accessor packages (when the `-a` option is given):

1. Accessor functions which return the localized formatted message for a message key. There are two accessor function packages generated:
 - A package for functions with return typed of `Wide_String`.
 - A package for functions with return typed of `String` (the standard Wide Strings are encoding using the encoding schema defined by the locale, the default being UTF-8 encoding). See *Encodings and The Codecs Type* for more information on encodings.

In both cases, the functions generated are named by using message key prefixed with the string `Format_`. This requires message keys, when prefixed with `Format_` be valid Ada identifier.
2. Accessor that print the localized formatted message similar to the standard `Put_Line` routines. There are two styles of routines generated:
 - Routines that use the standard `Ada.Wide_Text_IO` routines to print the formatted messages.
 - Routines that convert the `Wide_String` to `String` using the locale encoding and print this encoded string using Ada's `Stream_IO` routines.
3. Accessor procedures which raise an exception with a localized formatted message (similar to the standard `Raise_Exception` procedure but extended to allow message arguments). The procedures names generated prefix the message key with the string `Raise_`.

Normally, the narrow accessors should be used instead of the wide accessors as these support the user's encoding.

These accessor packages are generated for each facility compiled and are created as child packages of the command line package name.

As an example, reworking the moons example in terms of the safer accessor functions, the code would be, e.g., for the message that prints the number of moons for a planet, using the generated wide print routine:

```
with Messages.Moons_Wide_Prints;
...
Print_0002 (+Moons (Planet), +Planet));
```

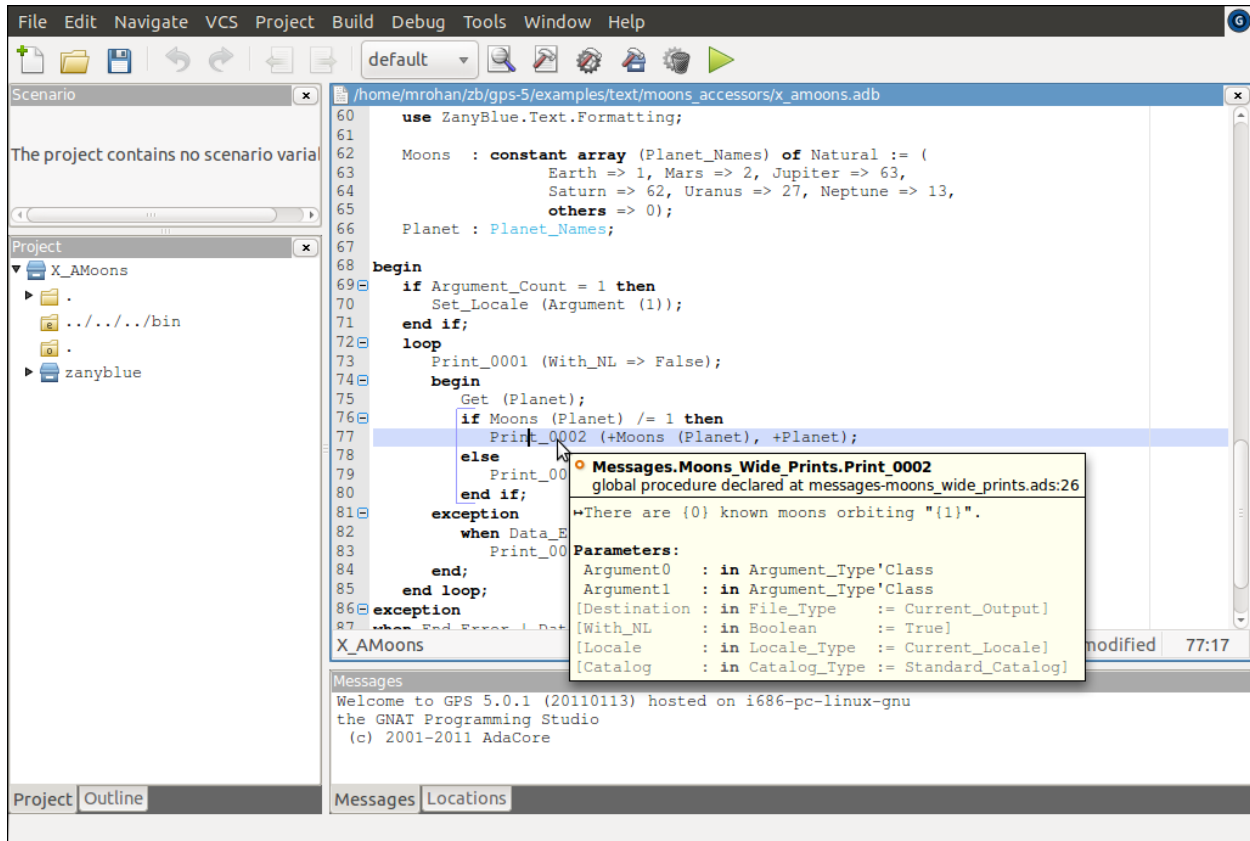
Here, the compiled messages are generated to the `Messages` package and the accessor packages are child packages of it.

For this accessor example, the `zbmcompile` utility must be used with the `-a` command line option to generate the accessor packages and the target parent package is given as `Messages` :

```
$ zbmcompile -a -i -v Messages Moons
This is ZBMCompile, Version 1.3.0 BETA (r3009M) on 6/20/16 at 1:59 AM
Copyright (c) 2009-2016, Michael Rohan. All rights reserved
Loaded 25 messages for the facility "Moons" (4 locales)
Performing consistency checks for the facility "Moons"
Performing consistency checks for the accessor package generation
Loaded 1 facilities, 7 keys, 4 locales and 25 messages
Loaded total 829 characters, stored 829 unique characters, 0% saving
Wrote the spec "Messages" to the file "./messages.ads"
Wrote the body "Messages" to the file "./messages.adb"
Generated accessor package spec "Messages.Moons_Exceptions" to "./messages-moons_
↳exceptions.ads"
Generated accessor package body "Messages.Moons_Exceptions" to "./messages-moons_
↳exceptions.adb"
Generated accessor package spec "Messages.Moons_Strings" to "./messages-moons_strings.
↳ads"
Generated accessor package body "Messages.Moons_Strings" to "./messages-moons_strings.
↳adb"
Generated accessor package spec "Messages.Moons_Wide_Strings" to "./messages-moons_
↳wide_strings.ads"
Generated accessor package body "Messages.Moons_Wide_Strings" to "./messages-moons_
↳wide_strings.adb"
Generated accessor package spec "Messages.Moons_Prints" to "./messages-moons_prints.
↳ads"
Generated accessor package body "Messages.Moons_Prints" to "./messages-moons_prints.
↳adb"
Generated accessor package spec "Messages.Moons_Wide_Prints" to "./messages-moons_
↳wide_prints.ads"
Generated accessor package body "Messages.Moons_Wide_Prints" to "./messages-moons_
↳wide_prints.adb"
ZBMCompile completed on 6/20/16 at 1:59 AM, elapsed time 0:00:00.124
```

The minor difference in the naming of the properties files should also be noted: for the example in the previous section, the files, and hence the facility, were all lower case. Here, the base file name is `Moons` . This allow the generated Ada packages to have the more expected name `Moons_Wide_Strings` .

Since the messages are now, in a sense, handled by the Ada compiler, IDE's, e.g., GPS, will display the message text (the generated packages include the message text, with markers explained later). The accessor display in GPS for the example message above is given in following image:



The keys used for the messages in this example are simple numeric style strings. The GPS IDE will also display the base language text and the number of arguments expected for accessors as the generated Ada code includes this text as a comment.

Message Formatting

From the simple `moons` example above, it can be seen that simply externalizing just the text used to in an application without the argument formatting is not enough to fully support localization. The messages externalized must be the complete messages, i.e., sentences, with embedded place holders for the arguments substituted at runtime.

The ZanyBlue Text library currently uses a mixture of Java and Python styles for embedded arguments. Arguments to the message are referenced by index (zero based) and enclosed in chain brackets.

From the `moons` example, message 0002 has the definition:

```
0002=There are {0} known moons orbiting "{1}".
```

Here the first argument (argument 0) is an integer and second argument (argument 1) is an enumeration value giving the planet name (but this is not explicitly defined in this message, see [Specifying Argument Types](#) for information on giving explicit type information).

At runtime, arguments to messages are “boxed” into a tagged type with dispatching methods that perform the formatting to strings. Each type has its own implementation (see [Argument Types](#)). Boxing occurs by creating a boxed object for the argument value and passed to the various message formatting or printing routines. The boxing function has a standard `Create` function and a renaming + making formatting or printing calls look more natural. For example, the message text above could be formatted with an integer argument, 2, and a string argument, Mars, as

```
Message : Wide_String := Format ("moons", "0002", +2, +String'("Mars"));
```

or, if the accessor package is used,

```
Message : Wide_String := Format_0002 (+2, +String'("Mars"));
```

The explicit type coercion to `String` is required as both standard `String` and `Wide_String` are supported.

The formatting or printing routines accept up to 5 optional boxed arguments. The implementation gathers the supplied boxed arguments into an argument list and then calls the underlying formatting routine with the argument list. For argument numbers beyond 5, the underlying argument list type must be used and the arguments must be explicitly appended, e.g., the above example could be rephrased in terms of the lower level argument list based formatting routine as

```
declare
    Arguments : Argument_List;
begin
    Arguments.Append (1);
    Arguments.Append (String'("Mars"));
    Display_Message (Format ("moons", "0002", Arguments));
end;
```

This 5 argument limit does *not* apply to the functions and procedures generated for accessor packages. These routines have arguments lists that match the number of expected arguments without the 5 argument limit.

Formatting Syntax

Additional formatting information can be included in the argument reference via an optional format string after the index value, separated by either a comma or a colon. The format syntax is based on the syntax defined for Python format strings in most cases (see the Argument Types section for details on types that use additional, non-Python style formatting, e.g, dates and time).

Specifying Argument Types

The format string is optionally prefixed by a type name separated by a comma character for the expected argument without space character (which would be interpreted as part for the format). E.g., to indicate a particular argument is should be an integer, the format would be:

```
0002=There are {0,integer} known moons orbiting "{1}".
```

The type names recognized are given in the following table:

Type Name	Description
any	No specific type required
boolean	Boolean values required
character	Character (wide or narrow) values required
date	A Calendar type required (formatted as a date)
datetime	A Calendar type required (formatted as a date and time)
duration	Duration type required
enum	Enumeration type required
exception	Exception type required (e.g., when E : others)
fixed	A fixed point value is required
float	A floating point value is required
integer	An integer value is required
modular	A modular type value is required
number	A numeric value (integer, real, etc) is required
real	A real value (float or fixed) is required
string	A string (wide or narrow) is required (characters also ok)
time	A Calendar type required (formatted as a time)

The type information, apart from the date and time related names, do not impact runtime formatting (the boxed value is formatted according to the boxed type formatting routine). The type information does, however, impact the signature of accessor routine generated.

For example, the message,:

```
0002=There are {0,integer} known moons orbiting "{1}".
```

would generate a format style access with the signature

```
function Format_0002 (
    Argument0    : Integer_Category_Type'Class;
    Argument1    : Any_Category_Type'Class;
    Locale       : Locale_Type := Current_Locale;
    Catalog      : Catalog_Type := Standard_Catalog) return Wide_String;
```

Here, argument 0 is required to be an integer type (verified by the compiler) while argument 1 is unconstrained with respect to type.

Use of the accessor allows the compiler to verify arguments have the expected type.

Type names have the expected hierarchy, e.g., a numeric argument type allows integer, float, fixed, etc, arguments.

General Formatting Syntax

The formatting information for the various types supported, in general, follow the Python/C style embedded formatting format for all but the date and time related formatting (the formatting of these values is described later in section *Date and Time Types*). E.g., to format an integer with a field width of 10 characters, the argument reference would be:

```
0001=Total number is {0,integer,10}
```

The syntax for the format information is (using standard BNF notation):

```
[[fill]align][sign][#][0][width][.precision][type][qual]
```

where

fill The fill character to use when padding, e.g., if the formatted value string is less than the field width additional padding characters are added fill out the field. Where the characters are added is defined by the next character, `align`. The fill character cannot be the `}` character.

align The align character defines the alignment of the formatted value within the minimum field width. The alignment characters recognized are

- < Value is flushed left, any additional padding characters needed are added to the right of the value.
- > Value is flushed right, any additional padding characters needed are prepend to the left of the value.
- = Any padding characters needed are added after the sign character, e.g., `+00010`. This is used only with numeric value. For non-numeric values, this alignment character is interpreted as `<`.
- ^ The value is centered in the field width with any padding character being added before and after the value to center it.

Note that unless a minimum field width is defined, the field width will always be the same size as the data to fill it, so the alignment option has no meaning in this case.

sign The sign character defines how the sign of numeric values are handled. This applies only to the sign of the value, the exponent of floating point numbers formatted in scientific notation is always preceeded by a sign character. It has three valid values:

- + A sign character is always generated. Positive values are preceeded by the locale plus character, negative values by the locale minus character.
- A sign character is only generated for negative values where the locale minus character is used. This is the default behaviour if no sign character is specified.
- “ “ (a space character) A sign character is only generated for negative values where the locale minus character is used, positive values are preceeded by a space character.

Note, the sign format character is ignored for non-numeric arguments.

The hash character causes integer formatted values formatted as binary, octal or hexadecimal to be decorated with the base using standard Ada notation. E.g., formatting the integer 2012 using base 16:

```
x => 7dc
#x => 16#7dc#
```

0 If the width field is preceded by a zero (0) character, this enables zero-padding. This is equivalent to an alignment type of `'='` and a fill character of `0`.

width The width is a Latin integer value defining the minimum field width for the argument. Padding, using the fill character, is added if needed to meet this minimum width.

precision The precision is a Latin integer value indicating how many digits should be displayed after the decimal point for a floating point value. The precision is not used for non-floating type formatting.

type The formatting style character gives the expected base to use when formatting integer arguments and style when formatting floating point arguments. Style indicators are ignored if the argument is not numeric. The integer formatting style characters supported are

- b** Binary format. Outputs the number in base 2.
- d** Decimal Integer. Outputs the number in base 10.
- o** Octal format. Outputs the number in base 8.
- x** Hex format. Outputs the number in base 16, using lowercase letters for the digits above 9.
- X** Hex format. Outputs the number in base 16, using uppercase letters for the digits above 9.

None The same as `d`.

The floating point formatting style characters supported are:

- E** Exponent notation. Prints the number in scientific notation using the localized equivalent of the letter E to indicate the exponent.
- e** Exponent notation. Same as E . Other formatting systems, e.g., C, would use case difference in the format string to change the case of the exponent character in the formatted value. Since localized versions are being used, it is not clear if lowercasing/uppercasing such strings is valid. The two format characters are treated the same.
- F** Fixed point. Displays the number as a fixed-point number.
- f** Fixed point. Same as F .
- G** General format. This prints the number as a fixed-point number, unless the number is too large, in which case it switches to E exponent notation.
- g** General format. Same as G .
- None** The same as E .

Infinity and NaN values are formatted as using localized versions.

qual The format string can be terminated with a final qualifier character. For the current version, the only valid value for this character is `*` which forces the formatting of the value using the Root locale, i.e., standard Ada Latin formatting. This impact the formatting of date and time values and the formatting of numbers where a localized version might use localized digits instead of the Latin 0123456789 , e.g., Arabic locales.

Argument Types

The formatting of messages with arguments is based on “boxing” message argument data values. The library provides a set of standard boxed types corresponding to the standard set of Ada types. Details on how the formatting these types is detailed in the following sections.

Boolean Type

Localized version of the English strings `true` and `false` are not available via the Unicode CLDR data. Boolean values are formatted as unlocalized English values. Examples, for format references are (the `*` is used to fill for readability):

Format	Result
Format ("{0}", +True)	"true"
Format ("{0,*>10}", +True)	"*****true"
Format ("{0,*<10}", +True)	"true*****"
Format ("{0,*^10}", +True)	"***true***"

Other formatting information is ignored, e.g., a formatting style character associated with numeric types, etc. The accessor argument type for Booleans is `Boolean_Category_Type` which is derived from the `Enumeration_Category_Type`.

Character Type

The character implementation simply inserts the character value as is to the formatted output. The library support both narrow and wide characters. Since `Create` exist for both, type information must be supplied if literal values are used. Field width, fill and alignment are respected, e.g., for the character variable `C` with a value of `a` :

Format	Result
Format ("{0}", +C)	"a"
Format ("{0, *>10}", +C)	"*****a"
Format ("{0, *<10}", +C)	"a*****"
Format ("{0, *^10}", +C)	"*****a*****"

Other formatting information is ignored, e.g., the format style character, etc. The accessor argument type for characters is `Character_Category_Type` which is derived from the `String_Category_Type`, i.e., a character is considered a “degenerate” string.

Duration Type

The standard implementation of the duration formatting displays the days, hours, minutes and seconds (seconds as a floating point formatted to three decimal places). If the number days is zero, it’s formatting is suppressed. The digits used are localized (this only impacts Arabic locales). Field width, fill and alignment are respected, e.g., for the Duration variable D

Format	Result
Format ("{0}", +D)	"16:48:03.000"
Format ("{0, *>16}", +D)	"****16:48:03.000"
Format ("{0, *<16}", +D)	"16:48:03.000****"
Format ("{0, *^16}", +D)	"**16:48:03.000**"

The accessor argument type for Durations is `Duration_Category_Type`.

Float Types

The `Float` and `Long_Float` formatting are simply instantiations of the `Generic_Floats` package, see [Generic Float Types](#) for more information. Some examples, details explained in the description of the generic package:

Format	Result
Format ("{0,e}", +Pi);	"3.14159E+00"
Format ("{0,f}", +Pi);	"3.141590"
Format ("{0,g}", +Pi);	"3.141590"
Format ("{0,.2f}", +Pi);	"3.14"

The accessor argument type for floats is `Float_Category_Type` which is derived from the `Real_Category_Type`.

Enumeration Types

Formatting of Ada enumeration types is handled via the generic package `Generic_Enumerations` which should be instantiated using the enumeration type. This allows enumeration values as arguments to messages. Obviously, no localization is available for the enumeration values (generated using `Wide_Image`). The generic package supplies a `Create` function to create the ZanyBlue “boxed” value along with a renaming for +.

Field width, fill and alignment are respected.

For example, for the definitions

```
type Colour is (Red, Green, Blue);
package Colour_Arguments is
  new ZanyBlue.Text.Generic_Enumerations (Colour);
use Colour_Arguments;
C : Colour := Red;
```

the example formatting is

Format	Result
<code>Format ("{0}", +C);</code>	<code>"RED"</code>
<code>Format ("{0,*<10}", +C);</code>	<code>"*****RED"</code>
<code>Format ("{0,*>10}", +C);</code>	<code>"RED*****"</code>
<code>Format ("{0,*^10}", +C);</code>	<code>"***RED***"</code>

The underlying implementation uses the `Image` function for the enumeration value and, as a result, is normally an uppercase value. The format width and placement (center, left, right) is respected. The accessor argument type for enumerations is `Enumeration_Category_Type`.

If localization of the enumeration image is required, the application will need to implement this. One possibility would be to create another facility that uses the enumeration image as a key and returns the localized value, e.g., for the `Colour` example above:

```
RED=Red
GREEN=Green
BLUE=Blue
```

and then use this facility to generate arguments, e.g.,

```
Format ("The colour is {0}",
        +Format ("colours", Colours'Wide_Image (C)));
```

There is, of course, a lot more work if the application were to support input of localized enumeration names.

Generic ZanyBlue text packages should be instantiated at the library level to prevent run time accessibility exceptions.

Generic Float Types

The `Generic_Floats` package implements formatting for floating point type. The formatting is based on David M. Gay's [\[Gay90\]](#) algorithm and attempts to produce accurate representations of floating point numbers (see also Guy Steele and Jon White's paper [\[Steele04\]](#)).

The formatting numeric style is controlled by the formatting style characters (unlike C, the case of the character does not matter):

- E** The floating point number is formatted using scientific notation, e.g., `1.23E+10`. Note in addition to the digits, the decimal point, sign characters and the exponent character are localized, e.g., in a Swedish locale the exponent characters is displayed as `times 10 to the power of`.
- F** The floating point number is formatted as a simple number. Note, for large absolute values of the exponent the formatted value will be a very long string of mainly zero characters.
- G** This format chooses the shorter of the `E` and `F` formatting depending on the value being formatted. For this release, the algorithm used is relative simplistic.

In addition to the type characters, the formatting width and precision are used when formatting floating point numbers:

width The total field width the formatted value should occupy. If the formatted value is smaller than this width, the result is padded to fill to this width (see the alignment characters later).

precision The precision is the number of digit displayed after the decimal point.

The plus or space character can be used to force either a plus or space character before the formatted number for positive values, negative numbers always include a sign character. The sign characters used are locale defined.

The alignment and fill characters are used to pad the result to the requested field width. In addition to the left, right and center alignment, floating point (and numeric values in general) also support the numeric alignment character =

which is simply a shorthand for align right using the `0` character for fill (this is the localized `0` character). However, to pad within base decorators, the `=` character must be used, e.g., to generate `16#003F#`.

As an example, the following table gives the various formatting options for the `Float` value `F := 2.9979E+8`:

Format	Result
<code>Format ("{0,e}", +F);</code>	<code>"2.99790E+08"</code>
<code>Format ("{0,f}", +F);</code>	<code>"299790000.0"</code>
<code>Format ("{0,g}", +F);</code>	<code>"2.99790E+08"</code>
<code>Format ("{0,.2f}", +F);</code>	<code>"299790000.00"</code>
<code>Format ("{0,.2e}", +F);</code>	<code>"3.00E+08"</code>
<code>Format ("{0,*>16e}", +F);</code>	<code>"*****2.99790E+08"</code>
<code>Format ("{0,=16e}", +F);</code>	<code>"000002.99790E+08"</code>

The implementation will use localized strings for infinity and *not a number* when formatting such values.

The accessor argument type for floats is `Float_Category_Type` which is derived from the `Real_Category_Type`.

Generic ZanyBlue text packages should be instantiated at the library level to prevent run time accessibility exceptions.

Generic Integer Types

The `Generic_Integer` package implements formatting for integer types (generic argument type `range <>`).

The formatting numeric style is controlled by the formatting style characters which control the base used when formatting, whether the value has a base decorator, handling of the sign, etc. The base selector characters are `begin{itemize}`

Style Character	Description
<code>b</code>	Format the value in binary (base 2)
<code>o</code>	Format the value in octal (base 8)
<code>d</code>	Format the value in decimal (base 10), this is the default
<code>x</code>	Format the value in hexadecimal (base 16), extra digits are the lower case Latin characters <code>a</code> to <code>f</code> . The CLDR data does not supply localized hexadecimal digits
<code>X</code>	Format the value in hexadecimal (base 16), extra digits are the upper case Latin characters <code>A</code> to <code>F</code>

If the format string includes the base decorator character `#` then the non-decimal format include the base, as per Ada syntax in the formatted result.

The alignment and fill characters are used to pad the result to the requested field width. In addition to the left, right and center alignment, integer (and numeric values in general) also support the numeric alignment character `=` which is simply a shorthand for align right using the `0` character for fill (this is the localized `0` character).

As an example, the following table gives the various formatting options for the `Integer` value `I := 42`:

Format	Result
<code>Format ("{0}", +I);</code>	<code>"42"</code>
<code>Format ("{0,b}", +I);</code>	<code>"101010"</code>
<code>Format ("{0,o}", +I);</code>	<code>"52"</code>
<code>Format ("{0,x}", +I);</code>	<code>"2a"</code>
<code>Format ("{0,X}", +I);</code>	<code>"2A"</code>
<code>Format ("{0,#b}", +I);</code>	<code>"2#101010#"</code>
<code>Format ("{0,#o}", +I);</code>	<code>"8#52#"</code>
<code>Format ("{0,#x}", +I);</code>	<code>"16#2a#"</code>
<code>Format ("{0,#X}", +I);</code>	<code>"16#2A#"</code>
<code>Format ("{0,=#10X}", +I);</code>	<code>"16#00002A#"</code>

The accessor argument type for integers is `Integer_Category_Type` which is derived from the `Number_Category_Type`.

Generic ZanyBlue text packages should be instantiated at the library level to prevent run time accessibility exceptions.

Generic Modular Types

The `Generic_Modulars` package implements formatting for modular types (generic type argument `mod <>`). This is essentially the same as the generic integers: the same rules apply, see *Generic Integer Types*.

The accessor argument type for integers is `Modular_Category_Type` which is derived from the `Integer_Category_Type`.

Generic ZanyBlue text packages should be instantiated at the library level to prevent run time accessibility exceptions.

Integer Type

This is simply an instantiation of the `Generic_Integers` package for the standard `Integer` type. See the generic integers description in section *Generic Integer Types*.

String Types

The string implementation simply inserts the string value as is to the formatted output. The library support both narrow and wide fixed and unbounded strings. Since `Create` exist for both, type information must be supplied if literal values are used. Field width, fill and alignment are respected, e.g., for the string variable `S` with a value of `abc`:

Format	Result
<code>Format ("{0}", +S)</code>	<code>"abc"</code>
<code>Format ("{0, *>10}", +S)</code>	<code>"*****abc"</code>
<code>Format ("{0, *<10}", +S)</code>	<code>"abc*****"</code>
<code>Format ("{0, *^10}", +S)</code>	<code>"*****abc***"</code>

Other formatting information is ignored, e.g., the type specifiers. The accessor argument type for characters is `String_Category_Type`.

Date and Time Types

The implementation for the formatting of times is the more involved and support two sub-categories to select either the time or date value of an `Ada.Calendar.Time` value. This is currently the only argument type that does not use the standard format string, e.g., you cannot specify a width, precision, etc. The root locale formatting is available, however, using a trailing `*` character in the format.

The built-in localization support includes localized formats for dates, times and date/times. These localization are implemented in terms of ASCII date/time format strings, e.g., the occurrence of the sequence `dd` within the format generated the day of the month in the output to as two characters (0 padded). The full set of format strings is documented in table below. (Note, some locale can have more than the simple am, noon, and pm for the day period, see *The zbinfo Utility* utility.) Characters not part of a recognized format substring are simply copied to the output as is. Sub-strings that should be included as is can be enclosed in single quotes (this is only needed if the sub-string would otherwise be interpreted as date/time values).

Time-Date Fromat	Description
a	Day period name, e.g., am , noon or pm
d	Day of the month, 1 .. 31
dd	Day of the month, 01 .. 31
EEEE	Full day of the week name
EEE	Abbreviated day of the week name
G	Era (CE/BCE, only CE is available with Ada)
h	Hours, 0 .. 12
HH	Hours, 00 .. 23
H	Hours, 0 .. 23
mm	Minutes, 00 .. 59
m	Minutes, 0 .. 59
MMMM	Full month name
MMM	Abreviated month name
MM	Month number 00 .. 12
M	Month number 0 .. 12
ss	Seconds, 00 .. 59
s	Seconds, 0 .. 59
YYYY	Full year, 2012, four digits
YY	Year, e.g., 12
y	Year, e.g., 2012, minimum number of digits
zzzz	Timezone names (not available, GMT offset printed)
z	GMT timezone offset printed

The date and time formatting is localized using the information from the CLDR data and include localized day and month names along with localized date and time formats.

The day in the week from a day is calculated using the code from [\[DayInWeek\]](#).

While a date/time format string can be included as part of the argument description, it is more normal to use the “pre-package”, locale aware, format styles:

short The basic information, e.g., a time format would likely not include the seconds. This is the default format.

medium Additional formatting on `short` , e.g., a time format would likely include the seconds, abbreviated month names in dates, etc.

long the additional formatting on `medium` , e.g., full month names, etc.

full the additional formatting on `long` , e.g., full day names, etc.

Using direct templates rather than the format styles might not work in all locales, i.e., generate mixed language results. E.g., if a template references the abbreviated month name, this will display as English in an Arabic locale (abbreviated month names do not appear to be used in Arabic).

The accessor argument type for characters is `Calendar_Category_Type` , all three format type strings (`date` , `time` and `datetime` map to this category type.

There is a lot of variety with date and times so the examples are more extensive that other types. In the following, the date/time being formatted is 4:48 pm, Blooms Day, June 16, 1904, referred to via the variable `B` . The examples below use two locales for demonstration purposes: `en_US` and `fr` .

Short date and times styles: default

The simplest use is to not specify anything. The generates an accessor using an `Any` type category, in this case the `datetime` is used to format. A type name is encouraged as the compiler will type check message arguments.

Format	Result
Format ("{0}", +B)	<ul style="list-style-type: none"> • en_US: "4:48 PM 6/16/04" • fr: "16:48 16/06/04"
Format ("{0,time}", +B)	<ul style="list-style-type: none"> • en_US "4:48 PM" • fr: "16:48"
Format ("{0,date}", +B)	<ul style="list-style-type: none"> • en_US: "6/16/04" • fr: "16/06/04"
Format ("{0,datetime}", +B)	<ul style="list-style-type: none"> • en_US: "4:48 PM 6/16/04" • fr: "16:48 16/06/04"

The formatting when no style is specified is short, e.g., `{0,time}` is the same as `{0,time,short}`.

Medium date and times styles

The medium style add more formatted values, e.g., using the example date. Due to space constraints, the formatting specifier is given rather than the full call to the `Format` function, e.g., `Format ("{0,time,long}", +B)`; is written as `time, long`.

Format	Result
time,medium	<ul style="list-style-type: none"> • en_US: "4:48:03 PM" • fr: "16:48:03"
date,medium	<ul style="list-style-type: none"> • en_US: "Jun 16,1904" • fr: "16 juin 1904"
datetime,medium	<ul style="list-style-type: none"> • en_US: "4:48:03 PM Jun 16,1904" • fr: "16:48:03 16 juin 1904"

Long date and times styles

The long style adds more formatted values on the medium style, e.g., again using the example date (example executed in the Pacific time zone, 7 hours earlier than GMT):

Format	Result
time, long	<ul style="list-style-type: none">• en_US: "4:48:03 PM -0700"• fr: "16:48:03 -0700"
date, long	<ul style="list-style-type: none">• en_US: "June 16, 1904"• fr: "16 juin 1904"
datetime, long	<ul style="list-style-type: none">• en_US: "4:48:03 PM -0700 June 16, 1904"• fr: "16:48:03 -0700 16 juin 1904"

Full date and times styles

Finally, the full style uses full day and month names, e.g., using the example date (example executed in the Pacific time zone, 7 hours earlier than GMT):

Format	Result
time, full	<ul style="list-style-type: none">• en_US: "4:48:03 PM -0700"• fr: "16:48:03 -0700"
date, full	<ul style="list-style-type: none">• en_US: "Thursday, June 16, 1904"• fr: "jeudi 16 juin 1904"
datetime, full	<ul style="list-style-type: none">• en_US: "4:48:03 PM -0700 Thursday, June 16, 1904"• fr: "16:48:03 -0700 jeudi 16 juin 1904"

Message Filtering

Frequently an application has different output modes, e.g., debug, verbose, normal, quiet, etc. If the various `Print` routines are used to generate messages for the application, these messages can be filtered using the tagged type `Message_Filter_Type` which is used by the ZanyBlue library to determine if a message should be really be printed.

The `Message_Filter_Type` defines the method

```
function Is_Filtered (Filter    : Message_Filter_Type;  
                     Facility  : Wide_String;  
                     Key       : Wide_String) return Boolean;
```

An application convention can be used on the message keys to define, e.g., verbose message filtering. The example application `examples/text/filtering` uses the convention:

- Verbose messages begin with the letter V
- Informational messages begin with the letter I
- Warning messages begin with the letter W

- Error messages begin with the letter E

The declaration of a filtering type for this configuration would be

```
type My_Filter_Type is new Message_Filter_Type with
  record
    Verbose : Boolean := False;
  end record;

function Is_Filtered (Filter   : My_Filter_Type;
                     Facility : Wide_String;
                     Key      : Wide_String) return Boolean;
```

with the simple implementation for this example being

```
function Is_Filtered (Filter   : My_Filter_Type;
                     Facility : Wide_String;
                     Key      : Wide_String) return Boolean is
begin
  return Key (Key'First) = 'V' and not Filter.Verbose;
end Is_Filtered;
```

To enable the filtering, the filter must be registered with the ZanyBlue library via the `Set_Filter` routine. This routine takes an access to `Message_Filter_Type`'Class object. E.g., for the example filtering application, the filter is installed using:

```
use ZanyBlue.Text.Formatting;
...
App_Filter : aliased My_Filter_Type;
...
Set_Filter (App_Filter'Access);
```

There is a cost associated with filtered messages:

- The various ZanyBlue routines are called.
- Any arguments will be “boxed” and appended to an argument list.
- The filtering code is called for all messages.

This cost, however, does not include the cost associated with formatting the message as the filtering occurs before any message formatting happens. Normally, it is this formatting cost that is the highest for general messages.

Stub Implementations

Some part of the ZanyBlue text library includes stub implementations. This section documents them.

Generic_Fixed

The `Generic_Fixed` package implemented formatting for the fixed floating point type. This is a stub implementation in this version of the ZanyBlue library and simply dispatches to the underlying Ada `Wide_Image` routine to format the value.

Generic ZanyBlue text packages should be instantiated at the library level to prevent run time accessibility exceptions.

ZanyBlue Formatting Implementation

The primary formatting method is the `Format` set of functions which format a message given a facility name, a key within that facility and a set of arguments (either as an `Argument_List` or as individual “boxed” arguments). The final two arguments for both this set of `Format` functions or the `Print` and `Print_Line` procedures (explained later) is the locale and the catalog. All the functions and procedures defined in this section are defined in the package `ZanyBlue.Text.Formatting` which is generally the only ZanyBlue package needed by applications.

The locale defaults to the current locale and generally need not be specified. A possible example of where a locale would need to be specified would be a client/server application where the client sends a locale name defining their preferred locale for messages.

The use of the catalog argument is even rarer and allows messages to be defined/loaded into separate catalogs. The ZanyBlue library maintains a global common catalog which is used as the default for all functions and procedures that take catalog arguments.

Format Functions

The specification of the `Format` function is

```
function Format (Facility  : Wide_String;
                Key       : Wide_String;
                Arguments : Argument_List;
                Locale    : Locale_Type := Current_Locale;
                Catalog   : Catalog_Type := Standard_Catalog)
return Wide_String;
```

along with the in-line boxed argument version:

```
function Format (Facility  : Wide_String;
                Key       : Wide_String;
                Argument0  : Argument_Type'Class := Null_Argument;
                Argument1  : Argument_Type'Class := Null_Argument;
                Argument2  : Argument_Type'Class := Null_Argument;
                Argument3  : Argument_Type'Class := Null_Argument;
                Argument4  : Argument_Type'Class := Null_Argument;
                Locale    : Locale_Type := Current_Locale;
                Catalog   : Catalog_Type := Standard_Catalog)
return Wide_String;
```

Print Procedures

Corresponding to the formatting functions, a set of `Print` and `Print_Line` procedures are available which print to the formatted message to the standard output file or the given file argument. These procedures have versions that take both an argument list, i.e.,

```
procedure Print (Facility  : Wide_String;
                Key       : Wide_String;
                Arguments : Argument_List;
                Locale    : Locale_Type := Current_Locale;
                Catalog   : Catalog_Type := Standard_Catalog);
```

and the in-line “boxed” arguments, e.g.,

```

procedure Print (Facility  : Wide_String;
                 Key       : Wide_String;
                 Argument0 : Argument_Type'Class := Null_Argument;
                 Argument1 : Argument_Type'Class := Null_Argument;
                 Argument2 : Argument_Type'Class := Null_Argument;
                 Argument3 : Argument_Type'Class := Null_Argument;
                 Argument4 : Argument_Type'Class := Null_Argument;
                 Locale    : Locale_Type := Current_Locale;
                 Catalog   : Catalog_Type := Standard_Catalog);

```

There are two options available when printing output to the `Current_Output` : use the `Text_IO.Current_Output` or `Wide_Text_IO.Current_Output`. The default uses the `Wide_Text_IO` version which defers any encoding to the Ada run-time (e.g., the GNAT `-gnatW8` command line option, etc). To use the `Text_IO` version, the `Formatting` routing `Disable_Wide_IO` should be used. This allows the ZanyBlue library to honour the encoding specified via the environment and is recommended, e.g.,

```

Disable_Wide_IO;
Print_Line ("xml", "0001", +My_Var);

```

The routine `Enable_Wide_IO` is available to re-enable the use of the `Wide_Text_IO` destination.

The signature for the `Print_Line` versions are similar. Both the `Print` and `Print_Line` procedure sets have corresponding versions that take a first argument giving the destination file, e.g.,

```

procedure Print (Destination : Ada.Wide_Text_IO.File_Type;
                 Facility    : Wide_String;
                 Key         : Wide_String;
                 Arguments   : Argument_List;
                 Locale      : Locale_Type := Current_Locale;
                 Catalog     : Catalog_Type := Standard_Catalog);

```

Plain Formatting Versions

Both the `Format` functions and `Print` procedure have versions that take a message format instead and arguments (either as an argument list or as in-line “boxed” arguments), e.g.,

```

function Format (Text      : Wide_String;
                 Arguments : Argument_List;
                 Locale    : Locale_Type := Current_Locale)
return Wide_String;

```

The locale argument is still required in this context as arguments are still formatted within the context of a locale.

Usage of these functions and procedures do not externalize the message text and, as such, do little to help internationalize applications.

Localized Exceptions

Ada allows exceptions to be raised with a message string, e.g.,

```

raise My_Exception with "Something is wrong here";

```

The ZanyBlue library includes `Raise_Exception` procedures with signatures paralleling the `Format` methods. The procedures raise the identified exception with a localized formatted messages. Since the Ada standard defines

exception message to be a `String`, the formatted `Wide_String` is converted to a `String` by UTF-8 encoding the `Wide_String`. The specification of the argument list version of this procedure is

```
procedure Raise_Exception (E      : Ada.Exceptions.Exception_Id;
                           Facility : Wide_String;
                           Key      : Wide_String;
                           Arguments : Argument_List;
                           Locale   : Locale_Type := Current_Locale;
                           Catalog  : Catalog_Type := Standard_Catalog);
```

The conversion of `Wide_String` to an UTF-8 encoded `String` uses the GNAT specific Unicode functions.

Missing Arguments and Exceptions

Format strings refer to arguments by index, e.g.:

```
moons=There are {0} moons orbiting "{1}".
```

expects two “boxed” arguments. If supplied with less than expected, e.g.,

```
Print_Line ("myapp", "moons", +10);
```

where the planet name is not supplied, is, by default, considered an error and the exception `No_Such_Argument_Error` is raised. This behavior can be adjusted by calling the catalogs routine `Disable_Exceptions`. When exceptions are disabled, missing arguments are replaced in the formatted string with the format information enclosed in vertical bars rather than braces.

The `Disable_Exceptions` has an inverse routine `Enable_Exceptions` which re-enables exceptions. This is either on the default standard catalog or a user supplied argument catalog. The status of exceptions for a catalog can be queried using the function `Exceptions_Enabled`.

Locale Type

The `Locale_Type` defines a locale which is used to select localized messages at run time. The type basically maps to the standard ISO names used to identify the language, script and territory for the locale. Typical examples of a locale name are

1. `fr` for French. Here only the language abbreviation is used. Here only the language is specified.
2. `en_US` for English as spoken in the United States, where the language and territory are specified.
3. `zh_Hant` for Traditional Chinese, where the language and script are specified.
4. `zh_Hans_CN` for Simplified Chinese as spoken in China where language, script and territory are specified.

Language and territory abbreviations must be either 2 or 3 characters in length, script abbreviations must be 4 characters. For a list the various language, script and territory codes, see the source properties files in `src/text/mesg`.

Locale Encodings

The system encoding name is normally appended to the this value separated by a period, e.g., `en_US.UTF-8`. The ZanyBlue library respects this encoding information and associates a `Codecs` object with the locale to encode and decode from and to `Wide_String` 's. The `Codecs_Type` associated with a locale is available via the `Codecs` method, e.g.,


```
Current_Codecs : constant Codecs_Type := Current_Locale.Codecs
```

See *Encodings and The Codecs Type* for more information on the `Codecs_Type`.

Locale Resolution

When accessing localized data, e.g., a message for a facility or some built-in localized data, the ZanyBlue library will perform the standard search through the locales for the data starting with the user supplied locale (normally the standard, current, locale).

This search applies rules to move from more specific locales to more general locales walking up a virtual tree of locales until the requested data is found or the root locale is reached.

The algorithm implemented in the ZanyBlue library is a general locale parenting algorithm which does the obvious for simple language and territory locales. E.g., the parent of the locale `de_DE` is `de` which, in turn, has as its parent the root locale. A similar algorithm is used for simple language and script locales, e.g., the parent of the locale `en_Latn` is `en`, which, again, has as its parent the root locale.

The locale parenting algorithm for full language, script and territory locales will have an ancestor tree of language and script, then language and territory, then language and finally the root locale. For example, the sequence of locales tried for the locale `"fr_Latn_FR"` is

1. `fr_Latn_FR`
2. `fr_Latn`
3. `fr_FR`
4. `fr`
5. Root Locale

This locale resolution occurs at run-time when a message for a particular facility is requested. The locale resolution for the built-in locale specific data, e.g., day names, time formats, etc., occurs at compile time. E.g., to access the name associated with Sunday requires only a few table lookups are at runtime, e.g.,

Function Call	Result
<code>Full_Day_Name (Make_Locale ("en"), Sun)</code>	"Sunday"
<code>Full_Day_Name (Make_Locale ("fr"), Mon)</code>	"lundi"
<code>Full_Day_Name (Make_Locale ("de"), Tue)</code>	"Dienstag"

Note: applications would normally just format, via message strings, values, e.g., `Time` values and let the type formatter access the lower level localized values, in this case when formatting `Time` values as dates, the localized date names might be accessed depending on the format style and locale.

It should also be noted the values returned for localized data are `Wide_Strings` and generally contain non-ASCII characters. Ad hoc testing on modern Unix (Linux) systems using X-Windows will display the correct characters (with the cooperation of the compiler, e.g., the `-gnatW8` switch for GNAT). On Windows, the DOS command will normally *not* display such character correctly. It is possible to enable display of non-ASCII characters via the selection of a code page for the command window.

Creating Locales

Values of `Locale_Type` are created via the `Make_Locale` function, e.g.,

```
My_Locale : constant Locale_Type := Make_Locale ("en_IE");
```

Changing Default Locale

The ZanyBlue Text routines allow the explicit definition of the locale for a particular function/procedure call but this normally not needed allowing the locale to default to the currently defined locale. The default locale is taken from the process environment via the variable `ZB_LANG`, and, if that is not defined, uses

1. On Unix systems, the value of the environment variable `LANG`.
2. On Windows systems, the translation of the user's default LCID value to a standard locale name (language, script and territory triple).

The default locale used can be adjusted at run time using the `Set_Locale` routine, e.g., to explicitly set the locale to Canadian French, the call would be

```
Set_Locale (Name => "fr_CA");
```

The Makefiles for the example applications generally include `run` targets which run the applications in the default locale. They also include rules to run application in other locales by tagging on the locale name to `run_`, e.g., to run the `texttt{tomcat}` example in a Greek locale, the command would be:

```
$ make run_el
```

Creating Locales

The default locale is created on application started and defaults to the locale associated with the running process. This is normally the expected locale. A locale can be explicitly created using the three variations of the `Make_Locale` function:

Simple String Locale Name

Creating a locale using a simple string contains the locale information of language, script, territory and encoding (not all are required, e.g.,:

```
en : constant Locale_Type := Make_Locale ("en");  
en_IE : constant Locale_Type := Make_Locale ("en_IE");  
en_Latn_IE : constant Locale_Type := Make_Locale ("en_Latn_IE");  
en_Latn_IE_ISO1 : constant Locale_Type := Make_Locale ("en_Latn_IE.ISO8859-1");
```

Message Source Locale

The current locale is used to locate localized messages in a facility, e.g., in a French locale, messages defined by the `_fr` properties files would be used, if available. If not available, then the messages defined by the base properties file will be used. If the application is developed in an English environment, then the base properties file will normally contain English messages. However, it is not uncommon to have non-English developers choose to use an English base properties file.

By default, the current locale is also used to format arguments to messages. This has the biggest impact for dates and times. For example, the following message:

```
Today=Today is {0,datetime,EEEE}.
```

will print the day name using the code

```
Print_Today (+Clock);
```

In an English locale, this will print the message:: Today is Thursday.

If the same application is run in a French locale and a French localization of the message is not available for the message, then the base message text is used, i.e., the English text. However, if the French locale will be used to format the date/time argument generating the message,:

```
Today is jeudi.
```

Such mixed language messages should normally be avoid in applications. It is better the entire message be in English, even in non-English locales.

To support this functionality, the ZanyBlue Text library associates a locale with each message. For localized messages, this is the locale associated with the localized property file, e.g., messages in the file `App_fr.properties` will have the locale `fr` associated with them. Using the associated message locale is controlled via the enable and disable source locale routines. This is enabled by default. See the source locale text example.

For the base message file, the default locale associated with the messages is the root locale. This should normally be explicitly set using the `zbmcompile -s` option.

Encodings and The Codecs Type

The codecs type captures the information related to encoding and decoding internal wide Unicode strings to externally useable character sequences, i.e., narrow strings. The current implementation initializes the locale based on the encoding defined by the `ZB_LANG` or `LANG` environment variable encoding definition. This is normally appended to the locale name separated by a period. Typical examples of `LANG` values are:

LANG	Description
<code>en_US.UTF-8</code>	English, as used in the United States with UTF-8 encoding
<code>fr_FR.ISO8859-1</code>	French, as used in France with ISO8859-1 encoding, Latin-1 encoding for Western European languages

Internally, the encodings are implementated as either direct, algorithmic, implementation, e.g., the UTF-8 encoding is implemented using the standard Ada runtime Unicode support packages, or as table driven lookups on the encoding translation data published on [\[UnicodeMappings\]](#).

The encodings support included, by default, in the ZanyBlue library are

Encoding	Description
ASCII	7-bit ASCII characters only
BIG5	Alias for CP950
CP874	ANSI/OEM Thai (ISO 8859-11); Thai (Windows)
CP932	ANSI/OEM Japanese; Japanese (Shift-JIS)
CP936	ANSI/OEM Simplified Chinese (PRC, Singapore)
CP949	ANSI/OEM Korean (Unified Hangul Code)
CP950	ANSI/OEM Traditional Chinese (Taiwan; Hong Kong SAR, PRC)
CP1250	ANSI Central European; Central European (Windows)
CP1251	ANSI Cyrillic; Cyrillic (Windows)
CP1252	ANSI Latin 1; Western European (Windows)
CP1253	ANSI Greek; Greek (Windows)
CP1254	ANSI Turkish; Turkish (Windows)
CP1255	ANSI Hebrew; Hebrew (Windows)
CP1256	ANSI Arabic; Arabic (Windows)

Continued on next page

Table 5.1 – continued from previous page

Encoding	Description
CP1257	ANSI Baltic; Baltic (Windows)
CP1258	ANSI/OEM Vietnamese; Vietnamese (Windows)
CP28591	Alias for ISO8859-1
CP28592	Alias for ISO8859-2
CP28593	Alias for ISO8859-3
CP28594	Alias for ISO8859-4
CP28595	Alias for ISO8859-5
CP28596	Alias for ISO8859-6
CP28597	Alias for ISO8859-7
CP28598	Alias for ISO8859-8
CP28599	Alias for ISO8859-9
CP28603	Alias for ISO8859-13
CP28605	Alias for ISO8859-15
CP65001	Alias for UTF-8
GB2312	Alias for CP936
ISO8859-1	ISO 8859-1 Western European
ISO8859-2	ISO 8859-2 Central European
ISO8859-3	ISO 8859-3 South European
ISO8859-4	ISO 8859-4 Baltic
ISO8859-5	ISO 8859-5 Cyrillic
ISO8859-6	ISO 8859-6 Arabic
ISO8859-7	ISO 8859-7 Greek
ISO8859-8	ISO 8859-8 Hebrew
ISO8859-9	ISO 8859-9 Turkish
ISO8859-10	ISO 8859-10 Nordic languages
ISO8859-11	ISO 8859-11 Thai
ISO8859-13	ISO 8859-13 Baltic Rim
ISO8859-14	ISO 8859-14 Celtic
ISO8859-15	ISO 8859-15 Western European (with Euro sign)
ISO8859-16	ISO 8859-16 South-Eastern European
SHIFT_JIS	Alias for CP932
UTF-8	Unicode 8-bit encoding

The `Codecs_Type` defines two functions to perform encoding and decoding on strings:

```
function Encode (Codecs : Codecs_Type;  
                Value  : Wide_String) return String;  
function Decode (Codecs : Codecs_Type;  
                Value  : String)   return Wide_String;
```

It is not generally necessary to the encoding functions as the formatting routines that return `String` 's encode using the `Codecs` associated with the locale. The decoding routine is not currently used by the library and is available if applications need to convert encoded input to `Wide_String` 's.

Built-in Localizations

Localization for dates, times, language names, script names and territory names are compiled into the ZanyBlue Text library based on the Unicode.org Common Locale Date Repository (CLDR). For details on date and time formatting see the section on `Ada.Calendar.Time` argument types later.

The localized name associated with standard language, script and territory abbreviations are available via the various routines defined in the package `ZanyBlue.Text.CLDR`.

The library is currently implemented to use the CLDR data to determine the zero character when printing numbers (integers). This is normally the standard ASCII “0”, however, some languages, e.g., Arabic, have their own numeric characters. In Arabic locale, `ZanyBlue.Text` will generate numeric (integer) output using Arabic numerals. Whether this is a feature or an error is unclear at this time.

For this release, the built-in locales are

Code	Language
ar	Arabic
cs	Czech
da	Danish
de	German
el	Greek
en	English
en_AU	English (Australia)
en_CA	English (Canada)
en_IE	English (Ireland)
en_GB	English (Great Britain)
en_NZ	English (New Zealand)
en_ZA	English (South Africa)
es	Spanish
fi	Finnish
fr	French
ga	Irish
he	Hebrew
hu	Hungarian
it	Italian
ja	Japanese
ko	Korean
nb	Norwegian Bokmål
nl	Dutch
pl	Polish
pt	Portuguese
ro	Romanian
ru	Russian
sk	Slovak
sv	Swedish
th	Thai
tr	Turkish
zh	Chinese (Simplified)
zh_Hant	Chinese (Traditional)

Running the example applications, in particular the `time` example, will display localized day, month, etc. names and localized formats.

Unicode CLDR Data

The Unicode.org CLDR data used to define the locale specific information such as the date and time formats also includes localized names for languages, scripts and territories. This localized information is included in the `ZanyBlue`

Text library via the `ZanyBlue.Text.CLDLDR` package and can be used to translate abbreviations, e.g. `en` to localized named, e.g., the function call,

```
Language_Name ("en")
```

returns `English` in an English locale and `anglais` in a French locale. There localized names for scripts and territories are available via the functions `Script_Name` and `Territory_Name` functions.

All functions take an optional `Unknown` parameter giving the result returned for unknown names (defaulting to the empty string) and a final locale parameter.

Pseudo Translations

One of the easiest mistakes to make with an internationalize application is to include hard-coded strings, i.e., not externalize the message text into a `.properties` file. One technique to detect hard-coded strings is to generate a pseudo translation in a test locale and test the application. This requires “translation” of a `.properties` file into a pseudo locale (the choice is normally Swahili in Kenya, i.e., `sw_KE`) and rebuild of a test application with the pseudo translations included.

ZanyBlue adopts a different approach and includes psuedo translation as part of the library rather than an after the fact exercise. The pseudo translation support built into the library support the translation of messages using simple wide character to wide character replacement, e.g., replace all ASCII character with their uppercase equivalents. Each message is further highlighted using start and end of message marker characters, the left and right diamond characters. Additionally, embedded arguments are surrounded by French quote characters.

To enable the built-in pseudo translations, the catalogs procedure

```
procedure Enable_Pseudo_Translations (Catalog : Catalog_Type;  
                                       Mapping : Pseudo_Map_Vector);
```

can be used. The `Mapping` argument gives the character to character mapping that should be used in addition to the message and argument marking of the pseudo translation.

The mappings defined by the ZanyBlue library are:

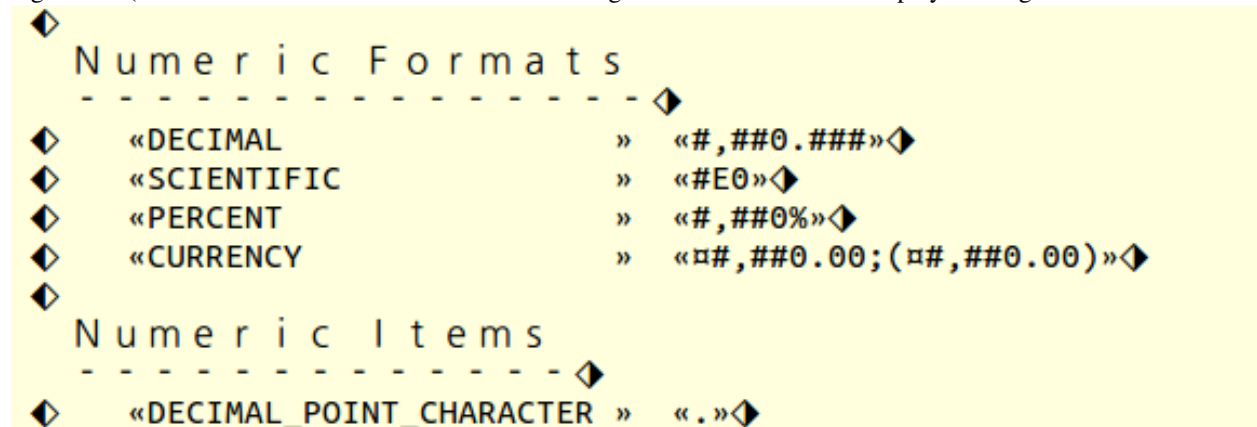
- `Null_Map` which preserves the message text but includes the start and end of messages and arguments.
- `Uppercase_Map` in addition to the start and end markers for messages and arguments, convert the message text to upper case (applies only to ASCII characters).
- `Lowercase_Map` in addition to the start and end markers for messages and arguments, convert the message text to lower case (applies only to ASCII characters).
- `Halfwidth_Forms_Map` in addition to the start and end markers for messages and arguments, convert the message text to the halfwidth forms for Latin alphabetic and numeric characters.
- `Enclosed_Alphanumeric_Map` in addition to the start and end markers for messages and arguments, convert the message text to the enclosed alphanumeric forms for Latin alphabetic characters.

Note: The halfwidth forms and enclosed alphanumeric mappings require the appropriate fonts be installed.

In addition to changing the characters used for the message, the Unicode character `Diamond with left half black` (U+2816) is prefixed and `Diamond with right half black` is suffixed. This allow the visual determination of where message strings begin and end. A relatively common programming error is to generate a message by concatenate a set of sub-messages. This is apparent in a psuedo translated view of the application.

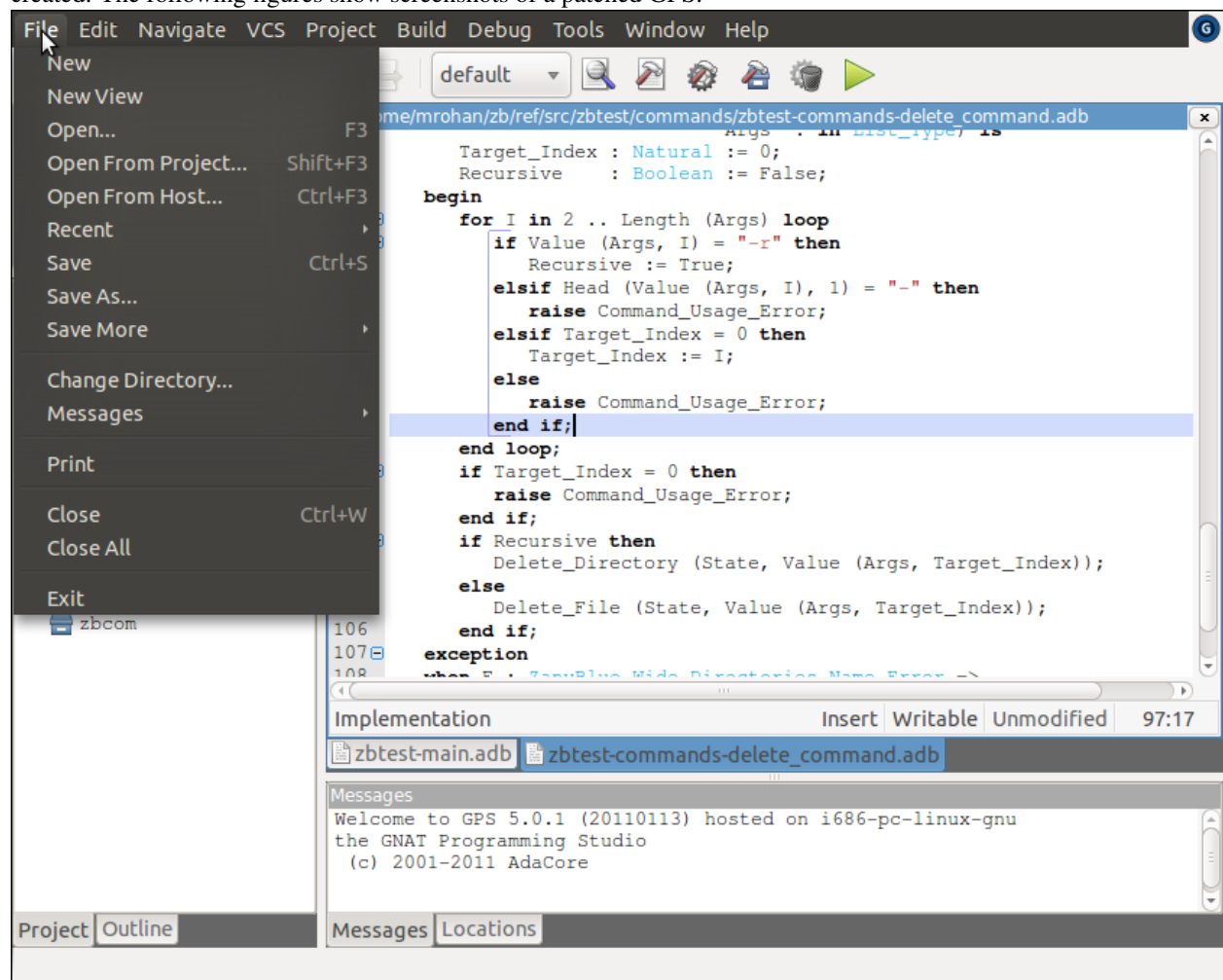
Normal argument handling occurs for pseudo translated messages and the values are substituted into the message string. The text of the values are not modified by psuedo translation. Value are, however, delimited by French quotes (guillemets, chevrons). The following figure shows the output of the `texttt{zbinfo}` example application with half width psuedo translation enabled. As can be seen from the message delimiters, the header `Numeric Formats` is

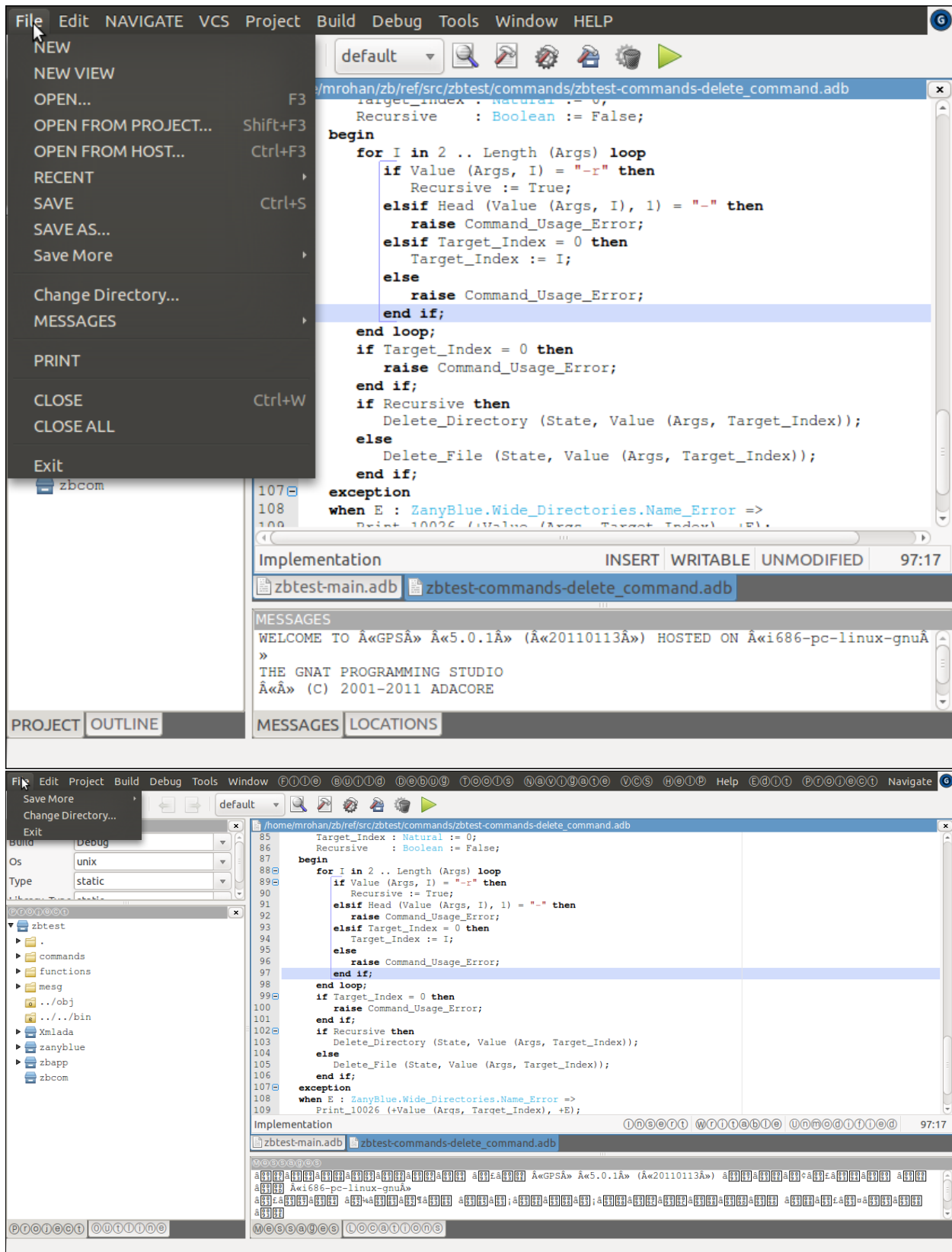
a multi-line messages and is displayed using the half width font. The Decimal, Scientific, etc, values are formatted arguments (as can be seen from the chevrons surrounding the values and are not displayed using the half width font.



The example applications support pseudo translation via the `verbl-xl` options.

The GPS example patches enable pseudo translation for GPS via the command line options `--pseudo= val`, where *val* is one of *u* (upper), *l* (lower), *h* (halfwidth) and *e* (enclosed). When halfwidth or enclosed mappings are used, the linkage between the standard menu item names and the localized names is lost and additional menu items are created. The following figures show screenshots of a patched GPS:





Message ‘Metrics’

Testing of internationalized applications (globalization, g11n, testing) is slightly different from normal functional testing. It is generally assumed the core functionality of the application is not dependent on the the messaging. Core testing is frequently driven by coverage data derived from the test cases. Performing testing in a globalized application to meet the same coverage to generally not something that is needed. The globalization testing needs to verify the application is fully localized in the target language and can handle localized input. This is generally a subset of the overall testing applied to an application.

To facilitate globalization testing, the library keeps track of the number of times a message is used giving a coverage number, from a message point of view. The accumulated usage information can be saved to an XML file using the `Write_Usage` routines in the text `Metrics` package.

The example application `moons_accessors` will generate a message usage report if given two command line arguments (the first being the locale name to use), e.g., for a simple execution with a usage file `moons.zbmx` given on the command line:

```
$ x_moons en moons.zbmx
This is MOONS, Version 1.3.0 - BETA
Please enter a planet: mars
There are 2 known moons orbiting "MARS".
Please enter a planet:
OK, goodbye.
```

The generate usage information looks like

```
<?xml version="1.0" encoding="utf-8"?>
<zanyblue-message-usage>
  <message facility="Moons" locale="de" key="0001" count="0" />
  <message facility="Moons" locale="" key="0003" count="1" />
  <message facility="Moons" locale="fr" key="0001" count="0" />
  <message facility="Moons" locale="de" key="0005" count="0" />
  <message facility="Moons" locale="" key="0004" count="0" />
  ...
  <message facility="Moons" locale="es" key="0001" count="0" />
  <message facility="Moons" locale="es" key="0003" count="0" />
</zanyblue-message-usage>
```

The contents of the file has been truncated for display purposes.

The zbmcompile Utility

The `zbmcompile` utility compiles `.properties` files into an Ada representation allowing easier access to the message text via lookup in a catalog.

The simplest usage of the utility defines the name of the package to be created and the facility to be compiled, e.g., for the `moons` example application:

```
$ zbmcompile -i -v Moons_Messages moons
This is ZBMCompile, Version 1.3.0 BETA (r3009M) on 6/20/16 at 8:50 AM
Copyright (c) 2009-2016, Michael Rohan. All rights reserved
Loaded 20 messages for the facility "moons" (4 locales)
Performing consistency checks for the facility "moons"
Loaded 1 facilities, 5 keys, 4 locales and 20 messages
Loaded total 677 characters, stored 677 unique characters, 0% saving
Wrote the spec "Moons_Messages" to the file "./moons_messages.ads"
```

```
Wrote the body "Moons_Messages" to the file "./moons_messages.adb"
ZBMCompile completed on 6/20/16 at 8:50 AM, elapsed time 0:00:00.033
```

Here the generated package `Moons_Messages` is compiled from the `.properties` files for the `moons` facility in the current directory.

The utility gives usage information when given the `-h` option:

```
$ zbmcompile -h
This is ZBMCompile, Version 1.3.0 BETA (r3059) on 8/6/16 at 7:44 PM
Copyright (c) 2009-2016, Michael Rohan. All rights reserved
Usage: zbmcompile [ options ] package facility
where available options are
  -q          Quiet mode
  -v          Verbose mode
  -A          Generate only ASCII source files
  -m          Always include argument modes in generated code
  -p          Number list entries (gnatcheck +RPositional_Components)
  -D          Debug mode (dump the catalog definitions)
  -d directory Directory containing facility (defaults to ".")
  -e extension Properties extension (default "properties")
  -h          Print this usage information
  -i          Include initialization call in body
  -O          Optimize string pool for locale based access
  -g          Disable locale based access optimization (default)
  -B          Compile only the base locale
  -u          Disable consistency checked (unchecked mode)
  -L locale   Compile only the named locale, e.g., "fr", "fr_FR"
  -r locale   Define reference (base) locale for consistency checking
  -s locale   Define source root locale
  -S stamp-file File to always create/update (contains a timestamp)
  -x name     Define external name for Initialize procedure
  -F          Force generation, ignoring errors. Generated code might
              not compile
  -a          Generate all accessors ([w]prints, [w]strings, exceptions)
  -G name     Generate named accessor ([w]prints, [w]strings, exceptions)
  -C          Disable comments on generated accessor routines
  -T item size Set the size of an Ada output limit: pool or comment
  -X handling Handling of non-Ada keys for accessors (error, ignore)
ZBMCompile completed on 8/6/16 at 7:44 PM, elapsed time 0:00:00.001
```

Controlling Status Output

The `zbmcompile` utility supports options to control the amount of status information printed:

- q** Reduced the amount of output to just error and warning messages.
- v** Increase the amount of output generated.
- D** Increase the amount of output to aid debugging.

Definition of Properties Directory

The default usage assumes the `.properties` files are located in the current directory. To locate the files in another directory, the `-d` option can be used, e.g.,:

```
$ zbmcompile -i -v -d msg Moons_Messages moons
```

would locate the properties associated with the `moons` facility in the `msg` directory.

Properties File Extension

The default file extension used when locating properties files is `.properties`. This can be change using the `-e` option. E.g., to load all `.msg` files,:

```
$ zbmcompile -i -v -e .msg Moons_Messages moons
```

Auto-Initialization

The generate Ada code include an initialization routine which loads the messages into a catalog (defaulting to the standard global catalog). The `-i` option includes a call to this initialization procedure in the body of the generated package. This allows the inclusion of the message in an application by simply including the specification in a compilation unit, normally the main unit. The option also causes the inclusion of a warning suppression pragma in the specification to allow compilation in a strict compilation environment.

Optimization of Messages

When the `zbmcompile` loads facilities in sequence which, in general, distributes the messages associated with various locales. The `zbmcompile` optimize mode, the `-O` option, performs a second pass on the loaded messages gathering messages for each locale together. Since applications generally don't change locale very often, if at all, having all the message strings for a locale located in the same set of pages can improve performance.

For symmetry reasons, the `-g` option is included which disables optimization.

Locale Selection

Occasionally, only a subset of the `.properties` files should be compiled into the generated Ada package. This selection is supported using the `-B`, for base locale, and `-L` options. E.g., to generate the Moons message package for just the base language, French and German, the command would be:

```
$ zbmcompile -v -B -L fr -L de Moons_Messages moons
```

These option could be used to generate language packs, possibly via shared library or dll implementations.

Forced Compilation

When testing using messages from Java projects, the message files will frequently be found to contain errors from a `verb|zbmcompile` point of view. To force the generation of Ada in the context of input errors, the `-F` can be used. Note, when used, there is no guarantee the resultant generated Ada code will compile.

Definition of External Initialization Routine

The possible direction for Ada localization is to allow the loading of language pack at run-time via shared library or dlls. This has not been investigated but in the context of dynamic loading of shared libraries or dll's, having an initialization name that is well defined makes the implementation easier. To support this, the untested functionality of supplying a linker name for the initialization routine is allowed via the `-x` option, e.g.,:

```
$ zbmcompile -v -B -x "moons_messages" Moons_Messages moons
```

Generation of Accessor Packages

The generation of message accessor packages creating routines for each message defined with parameter lists matching the arguments defined by the message text is controlled via the two options `-a` and `-G`.

The first style, `-a`, generates all accessor style packages:

1. `exceptions`, generate routines to raise exceptions with localized message strings (wide strings converted to encoding associated with the locale).
2. `strings`, generate functions returning locale encoded strings for the localized messages.
3. `wstrings`, generate functions returning wide strings for the localized messages.
4. `prints`, generate routines printing the localized messages to files as locale encoded strings.
5. `wprints`, generate routines printing the localized messages to files as wide strings (wide files).

The `-G` option allows the selection of individual accessor style packages, e.g., for an application that only uses messages to raise exceptions and print to wide files, the command line options would be `-G exceptions` and `-G wprints`, i.e., the `-G` option can be used multiple times on the same command line.

The packages generated are child packages of the primary package given on the command line with names based on the facility name, e.g., if the facility name is “Moons”, the generated child packages would be

Style	Child Package Name
<code>exceptions</code>	<code>Moons_Exceptions</code>
<code>strings</code>	<code>Moons_Strings</code>
<code>wstrings</code>	<code>Moons_Wide_Strings</code>
<code>prints</code>	<code>Moons_Prints</code>
<code>wprints</code>	<code>Moons_Wide_Prints</code>

Handling non-Ada Message Keys

When generating accessors, the keys in the various properties files are assumed to be valid Ada identifiers. `zbmcompile` raises an error if it encounters non-Ada identifier keys, e.g., keys containing periods.

The command line option `-X` can be used to define how such keys should be handled with values of `error` to generate errors for such keys and abandon the generation process (this is the default behaviour), or `ignore` to simply ignore such keys and continue generating accessors for the valid Ada identifier keys.

Output Directory

By default, the generated packages are written to the current directory. To select a different directory, the `-o` option can be used, e.g.,:

```
$ zbmcompile -o msg Moons_Messages moons
```

would write the packages to the `msg` directory. The directory must already exist.

Adjusting the Generated Code

The `zbmcompile` command has a number of options used to adjust the generated code.

Comments for Accessors

By default, the generated routines for accessors include the base message text as a comment. This allows the display of the text of the messages within GPS and makes browsing the source easier. These comments can be suppressed using the `-C` option. One reason to suppress these comments would be to minimize recompilations when updating messages. The `zbmcompile` command will only create new source files if the generated contents differs from the existing files (or the files currently doesn't exist). With comments suppressed, updating message strings would only result in the update to the primary package body requiring, in general, a single recompilation and re-link of an application. With comment enabled, the accessor spec files would be updated resulting in larger recompilations.

Argument Modes

The default code generated does not include the `in` keyword for in routine arguments. Some code bases might have style rules requiring explicit use of this keyword. To require the generated code include this keyword, the `-m` option can be used.

Positional Elements

The generated code includes a number of initialized tables (arrays). The default style for such tables is simply to list the entries using implicit index association. Again, some code bases might require that explicit numbering of such code. This can be enabled using the `-p` command line option. E.g., instead of

```
Facilities : constant ZT.Constant_String_List (1 .. 7) := (
    Facility_1'Access,
    Facility_2'Access,
    ...
```

the generated code would be

```
Facilities : constant ZT.Constant_String_List (1 .. 7) := (
    1 => Facility_1'Access,
    2 => Facility_2'Access,
    ...
```

Output Line Lengths

The generated code keeps within the standard 80 column style for source files. There are two control parameters which can be used as arguments to the `-T` command line option to adjust this for selected items:

1. The accumulated message strings are stored as a single constant string initialized using a multi-line string. The length of the substrings written per line can be controlled using the command line `pool` item with an integer value, e.g., `-T pool 30` to reduce the size.
2. The base message text written as a comment on accessors is wrapped to ensure the 80 column limit is not exceeded. This results in wrapping within words. To increase the limit for messages, use the `comment` item, e.g., `-T comment 120`. Accessor comments include breaks for messages with new lines.

Consistency Checks

Localized messages are cross checked with the base locale messages to ensure they are consistent, i.e., it is an error for a localized message to refer to an argument not present in the base message.

There are two options that control the consistency checks performed

1. `-u` disable the consistency checks. This is a rather dangerous thing to do and should be avoided.
2. `-r` define the base locale. Normally the base locale messages are in the properties file without a locale, i.e., the root locale. Some applications might choose to use explicit locale naming for all properties files. The `-r` option can be used to designate which of the available locales is the base locale.

Selection of Source Locale

The source locale for the base properties file can be specified using the `-s` option. See [Message Source Locale](#) for details on this functionality.

Generating ASCII Only Sources

By default, the `zbmcompile` utility generates UTF-8 encoded source files, in particular, the `Wide_String` used to store the compiled messages is a UTF-8 encoded string in the source file.

It is assumed the compilation environment will be configured to use UTF-8 encoded sources, e.g., the GNAT `-gnatW8` compilation option. The `zbmcompile` command line option `-A` causes the generated source files to use string concatenation along with `Wide_Character` 'Val' for non-ASCII characters in these strings.

Stamp File

For Makefile base builds, dependency checking is simplified if a single fixed named file can be used rather than the set of generated spec and body files normally created by `zbmcompile`. The `-S` option allows the definition of a simple time stamp file which is updated whenever the `zbmcompile` command is run. This file can be used to define built dependencies in Makefiles.

Command Line Utilities

The command line utilities available are generally covered in other areas of the documentation. The full documentation for some general utilities is given here, the other utilities will simply refer back to the section that covers them.

The `zbinfo` Utility

The `zbinfo` utility prints information on data built into the ZanyBlue library. The utility usage is

```
$ zbinfo -h
This is ZBInfo, Version 1.3.0 BETA (r3059) on 8/6/16 at 7:44 PM
Copyright (c) 2016, Michael Rohan. All rights reserved
The "zbinfo" command prints information on the data built into ZanyBlue
library. By default, information on the current locale is printed. The
available options are:

-h|--help          Print this help information
-r|--reverse       Reverse the encoding dump (map encoded to Unicode)
--list-encodings   List the available encoding schemes
--dump-encoding name Print information on the named encoding
--dump-locale name Print information on the named locale

ZBInfo completed on 8/6/16 at 7:44 PM, elapsed time 0:00:00.001
```

The zbmcompile Utility

The `zbmcompile` utility compiles message properties files into Ada sources for use in applications. It is described in the section *The zbmcompile Utility*. It's usage is

```
$ zbmcompile -h
This is ZBMCompile, Version 1.3.0 BETA (r3059) on 8/6/16 at 7:44 PM
Copyright (c) 2009-2016, Michael Rohan. All rights reserved
Usage: zbmcompile [ options ] package facility
where available options are
-q          Quiet mode
-v          Verbose mode
-A          Generate only ASCII source files
-m          Always include argument modes in generated code
-p          Number list entries (gnatcheck +RPositional_Components)
-D          Debug mode (dump the catalog definitions)
-d directory Directory containing facility (defaults to ".")
-e extension Properties extension (default "properties")
-h          Print this usage information
-i          Include initialization call in body
-O          Optimize string pool for locale based access
-g          Disable locale based access optimization (default)
-B          Compile only the base locale
-u          Disable consistency checked (unchecked mode)
-L locale   Compile only the named locale, e.g., "fr", "fr_FR"
-r locale   Define reference (base) locale for consistency checking
-s locale   Define source root locale
-S stamp-file File to always create/update (contains a timestamp)
-x name     Define external name for Initialize procedure
-F          Force generation, ignoring errors. Generated code might
           not compile
-a          Generate all accessors ([w]prints, [w]strings, exceptions)
-G name     Generate named accessor ([w]prints, [w]strings, exceptions)
-C          Disable comments on generated accessor routines
-T item size Set the size of an Ada output limit: pool or comment
-X handling Handling of non-Ada keys for accessors (error, ignore)
ZBMCompile completed on 8/6/16 at 7:44 PM, elapsed time 0:00:00.001
```

The zbtest Utility

The `zbtest` utility is a simple application testing utility which compares expected output (with regex masking) for a command with the output actually generated. It has a hierarchical test script structure based on the testing directory structure. At this point, the only documentation of this utility is the source and the test scripts in the `src/test/system` directory. It's usage is

```
$ zbtest -h
This is ZBTest, Version 1.3.0 BETA (r3059) on 8/6/16 at 7:44 PM
Copyright (c) 2009-2016, Michael Rohan. All rights reserved
zbtest [ -h ] [ -t test-area ] [ -X xml-summary ] [ -d param value ] [ test-script ]
ZBTest completed on 8/6/16 at 7:44 PM, elapsed time 0:00:00.003
```

Additional Documentation

In addition to the documentation here, the documentation generated from the Ada spec source files via GNATDOC is also available. This is currently the only documentation available for the non-Text packages, e.g., the parameters handling, etc.

Contributions

This software includes contributions from the following open source developers:

- *Pascal Pignard*

Contributions to ZanyBlue are contributed under the same BSD style license that covers the ZanyBlue project. The following sections document the contributions made to the ZanyBlue project.

Pascal Pignard

Contact: blady-com@users.sf.net.

Feb, 2016 The initial ISO8859-2 implementation of the Encoding/Decoding support was implemented and contributed by Pascal.

Notices

ZanyBlue software includes code and data from the following open source projects:

- *Ahven, 2.6*
- *Apache Tomcat, 9.0.0.M8*
- *CLDR, release-26-0-1*
- *Jenkins CI, jenkins-2.9*
- *XML/Ada, 2016*

The conditions for the use is detailed below.

Ahven, 2.6

This project uses the Ahven framework for unit testing.

The copyright notice for this project is:

```
-- Ahven Unit Test Library - License
--
-- Copyright (c) 2007-2015 Tero Koskinen <tero.koskinen@iki.fi>
--
-- Permission to use, copy, modify, and distribute this software for any
-- purpose with or without fee is hereby granted, provided that the above
-- copyright notice and this permission notice appear in all copies.
--
-- THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
-- WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
```



```
-- MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
-- ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
-- WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
-- ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
-- OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
--
```

Apache Tomcat, 9.0.0.M8

Some of the example properties files were taken from the Apache Tomcat project.

The copyright notice for this project is:

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

CLDR, release-26-0-1

The built-in localizations, language name, date/time formats, etc, are generated from from the Unicode.org's Common Locale Data Repository.

The copyright notice for this project is:

```
EXHIBIT 1
UNICODE, INC. LICENSE AGREEMENT - DATA FILES AND SOFTWARE

Unicode Data Files include all data files under the directories
http://www.unicode.org/Public/, http://www.unicode.org/reports/,
and http://www.unicode.org/cldr/data/. Unicode Data Files
do not include PDF online code charts under the directory
http://www.unicode.org/Public/. Software includes any source
code published in the Unicode Standard or under the directories
http://www.unicode.org/Public/, http://www.unicode.org/reports/, and
http://www.unicode.org/cldr/data/.

NOTICE TO USER: Carefully read the following legal agreement. BY
DOWNLOADING, INSTALLING, COPYING OR OTHERWISE USING UNICODE INC.'S DATA
FILES ("DATA FILES"), AND/OR SOFTWARE ("SOFTWARE"), YOU UNEQUIVOCALLY
ACCEPT, AND AGREE TO BE BOUND BY, ALL OF THE TERMS AND CONDITIONS OF
THIS AGREEMENT. IF YOU DO NOT AGREE, DO NOT DOWNLOAD, INSTALL, COPY,
DISTRIBUTE OR USE THE DATA FILES OR SOFTWARE.

COPYRIGHT AND PERMISSION NOTICE
```

Copyright (c) 1991-2011 Unicode, Inc. All rights reserved. Distributed under the Terms of Use in <http://www.unicode.org/copyright.html>.

Permission is hereby granted, free of charge, to any person obtaining a copy of the Unicode data files and any associated documentation (the "Data Files") or Unicode software and any associated documentation (the "Software") to deal in the Data Files or Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Data Files or Software, and to permit persons to whom the Data Files or Software are furnished to do so, provided that (a) the above copyright notice(s) and this permission notice appear with all copies of the Data Files or Software, (b) both the above copyright notice(s) and this permission notice appear in associated documentation, and (c) there is clear notice in each modified Data File or in the Software as well as in the documentation associated with the Data File(s) or Software that the data or software has been modified.

THE DATA FILES AND SOFTWARE ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE DATA FILES OR SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in these Data Files or Software without prior written authorization of the copyright holder.

Unicode and the Unicode logo are trademarks of Unicode, Inc. in the United States and other countries. All third party trademarks referenced herein are the property of their respective owners.

Jenkins CI, jenkins-2.9

Some of the example properties files were taken from the Jenkins project.

English

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2010, Sun Microsystems, Inc., Kohsuke Kawaguchi, id:cactusman, ↵
↵Seiji Sogabe
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
```

```
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Bulgarian

The copyright notice for this is:

```
# The MIT License
#
# Bulgarian translation copyright 2015 Alexander Shopov <ash@kambanaria.org>.
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Danish

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2010, Sun Microsystems, Inc. Kohsuke Kawaguchi. Knud Poulsen.
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
```

```
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

German

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2009, Sun Microsystems, Inc., Kohsuke Kawaguchi, Simon Wiest
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Spanish

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2009, Sun Microsystems, Inc., Kohsuke Kawaguchi, id:cactusman
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
```

```
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

French

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2009, Sun Microsystems, Inc., Kohsuke Kawaguchi, Eric Lefevre-
# →Ardant
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Italian

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2010, Sun Microsystems, Inc., Kohsuke Kawaguchi, Giulio D'Ambrosi
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
```

```
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Japanese

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2010, Sun Microsystems, Inc., Kohsuke Kawaguchi, Seiji Sogabe, ↵
↵id:cactusman
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Dutch

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2009, Sun Microsystems, Inc., Kohsuke Kawaguchi, id:sorokh
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Polish

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2016, Damian Szczepanik
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Portuguese

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-, Kohsuke Kawaguchi, Sun Microsystems, Inc., and a number of
↳ other of contributors
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Brazilian Portuguese

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2009, Sun Microsystems, Inc., Kohsuke Kawaguchi, Reginaldo L.
↳Russinholi, Cleiber Silva, Fernando Boaglio
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Russian

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2009, Sun Microsystems, Inc., Kohsuke Kawaguchi, Mike Salnikov
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Turkish

The copyright notice for this is:


```
# The MIT License
#
# Copyright (c) 2004-2009, Sun Microsystems, Inc., Kohsuke Kawaguchi, Oguz Dag
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

Simplified Chinese

The copyright notice for this is:

```
# The MIT License
#
# Copyright (c) 2004-2010, Sun Microsystems, Inc., Kohsuke Kawaguchi, id:cactusman, ↵
↵Seiji Sogabe
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
```

XML/Ada, 2016

This project uses GNAT's XML/Ada to generate the Jenkins compatible test reports for the zbtest utility.

The copyright notice for this project is:

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for
software and other kinds of works.

The licenses for most software and other practical works are designed
to take away your freedom to share and change the works. By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users. We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors. You can apply it to
your programs, too.

When we speak of free software, we are referring to freedom, not
price. Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights. Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received. You must make sure that they, too, receive
or can get the source code. And you must show them these terms so they
know their rights.

Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software. For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so. This is fundamentally incompatible with the aim of
protecting users' freedom to change the software. The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable. Therefore, we
have designed this version of the GPL to prohibit the practice for those
products. If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions

of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your

work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no

charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent

that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may

not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is

conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's

public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by

the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License. Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program
into proprietary programs. If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library. If this is what you want to do, use the GNU Lesser General
Public License instead of this License. But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.

References

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

- [Steele04] Guy L. Steele Jr. and Jon L. White, ACM SIGPLAN Notices, Vol 39, No 4, April, 2004. “How to Print Floating-Point Numbers Accurately”, [ACM Portal](#).
- [Burger96] Robert G. Burger and R. Kent Dybvig, Proc. ACM SIGPLAN ‘96 Conf. Prog. Lang. Design and Implementation, “Printing Floating-Point Numbers Quickly and Accurately”, June, 1996, ACM (Philadelphia, PA), [PDF Download](#).
- [Gay90] David M. Gay, “Correctly Rounded Binary-Decimal and Decimal-Binary Conversions”, Nov, 1990, AT&T Bell Laboratories, Murray Hill, NJ, [Postscript](#).
- [DayInWeek] Calculating the day of the week, [Wikipedia](#).
- [UnicodeMappings] <http://www.unicode.org/Public/MAPPINGS>