

# Globus Toolkit Localization Guidelines

**PSV-001-0002**

Revision: 2.0

3 October, 2005

Document Status: **Final**

Patrick Doran  
Robert C. Gaffaney

Copyright 2005 Univa Corporation

Licensed 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.

**Disclaimer:** The electronic copy is the official copy and supersedes any printed copy.

Key words: Globus, Internationalization, Localization

***This project was funded by the Globus Consortium, Inc.***

## TABLE OF CONTENTS

1.	REVISION HISTORY .....	3
2.	RELATED DOCUMENTS AND LINKS.....	4
3.	INTRODUCTION.....	5
3.1	Glossary of Terms.....	5
3.2	Purpose .....	7
4.	OVERVIEW OF GLOBUS TOOLKIT L10N .....	9
5.	HARDWARE AND SOFTWARE REQUIREMENTS FOR L10N AND TESTING .....	12
5.1	Localization Support Files to Be Distributed with the Globus Toolkit.....	12
5.2	Hardware and Software Requirements for Globus Developers .....	12
5.3	Hardware and Software Requirements for Localization .....	13
6.	L10N RESOURCES .....	14
7.	TESTED LOCALES .....	15
8.	LOCALIZATION GUIDELINES .....	16
8.1	Localizing Globus Toolkit Resources .....	16
8.1.1	Source Management During Localization.....	16
8.1.2	Installing Globus Toolkit and Testing Translated Resources .....	16
8.1.3	Localizing the Java ListResourceBundles.....	16
8.1.4	Localizing the ICU Resource Files .....	19
8.1.5	Localizing Script Resource Files .....	25
8.1.6	Localizing Globus Toolkit User Documentation .....	25
8.2	Verifying and Packaging Localized Resources .....	25
8.2.1	Running the I18N Resource Tests .....	25
8.2.2	Merging Localized Resources into the Mainline .....	26
8.2.3	Formal Validation of Localized Resources .....	26
8.2.4	Packaging Localized Content for Release.....	27
9.	SUPPORT .....	28

## **1. REVISION HISTORY**

Revision 0.1	May 15, 2005	Patrick Doran	Initial Entry
Revision 0.2	July 7, 2005	Patrick Doran	Internal Review
Revision 1.0	August 17, 2005	Patrick Doran	Published for Review
Revision 2.0	October 3, 2005	Patrick Doran	Final, Revised per Review

## **2. RELATED DOCUMENTS AND LINKS**

[EDS-1003] Globus Toolkit Globalization Coding Guidelines

[GT] Globus Toolkit – The Globus Alliance [www.globus.org](http://www.globus.org)

[DEIT-2001] Deitch, Andrew; Czarnecki, David: Java Internationalization O'Reilly, March 2001.

[ICU-UG] ICU User Guide – <http://icu.sourceforge.net/userguide/>

[ICU-API] ICU API Documentation – <http://icu.sourceforge.net/apiref/>

### 3. INTRODUCTION

Univa Corporation was contracted by the Globus Consortium to perform work related to Globalization of the Globus Toolkit. Note that this project will not implement the roadmap defined. Timing and tasking of the Implementation will be determined at a later date.

The output of this contract is in the form of three documents to be published at its completion:

- Globalization Guidelines: A comprehensive description of the strategy, tools and procedures to be used in Globalization.
- Localization Guidelines: This document
- Globalization Scoping Assessment: An estimate of the size of the Globalization effort given the Globus Toolkit code base and the Guidelines developed.

This document provides instructions for localization (L10N) of a Globus Toolkit software release. We provide an overview of the L10N process, documenting the workflow for the Globus Developers and L10N contributors from completion of coding on a Globus Toolkit release through L10N of Globus Toolkit resources and testing and integration of localized resources into the Globus Toolkit release. This document does not cover internationalization (I18N) of Globus Toolkit code. For information on Globus Toolkit I18N, see [EDS-1003].

#### 3.1 Glossary of Terms

Term	Definition
Bi-Directional (Bidi) Languages	Different languages use different directions for text display. Bi-directional languages allow text to be in multiple directions. For example, most Arabic and Hebrew text is written right to left, but numbers and some borrowed words are written from left to right. This means software supporting bi-directional languages cannot assume one logical character order when processing text.
Globus Toolkit Distribution	The Globus Toolkit distribution including all Globus Toolkit core component, applicable Commodity Grid (CoG) Kits, and included libraries and tools.
Globalization (G11N)	G11N is a blanket term that covers both I18N and L10N. Globalized software is internationalized, so locale specific resources can be translated without recompiling. In addition, globalized software has been localized for all user locales, so the same software combined with local specific resources can provide an appropriate interface

<b>Term</b>	<b>Definition</b>
	for users in other locales.
Globus Contributor	Globus Contributors include Globus community members outside the Globus Alliance who make enhancement to the Globus Toolkit. Rather than submitting enhancement directly in the CVS repository, Globus Contributors submit enhancements as contributions to the toolkit, and Globus Alliance Developers review the contributions and merge them into the Globus Toolkit.
Globus Toolkit Repository	The CVS repositories used to manage the source code and supporting libraries for the Globus Toolkit, including the separate repository used to manage source code for Commodity Grid (CoG) components.
Globus Developer	Globus Developers include all Globus Alliance Developers assigned to work on the Globus Toolkit. Globus Developers submit code directly to the Globus CVS Repository.
ICU, ICU4C, ICU4J	The International Components for Unicode (ICU) was developed by IBM to provided enhanced Unicode and I18N support for C and Java applications. Classes originally developed for ICU for Java (ICU4J) were incorporated into the Core Java 1.1 SDK, and the current ICU4J implementation includes enhanced versions of some of these classes, plus additional classes that complement the classes in the JDK. C and C++ versions of the same international functionality are available in ICU for C (ICU4C). The ICU4J and ICU4C APIs differ slightly due to language differences and new functionality. For example, ICU4C includes a character converter API.
Internationalization (I18N)	Updating software so locale specific resources such as strings in the UI can be translated and provided to users without needing to recompile the executable.
Locale	A locale is a combination of a target language and a specific cultural setting. For example, Spanish – Iberia would be a different locale that Spanish – Mexico.

Term	Definition
Localization (L10N)	The process of adapting internationalized software for use in a given language and locale. This typically involves translating externalized strings and graphics and creating locale specific resource bundles.
Resource Bundle	<p>In Java, resource bundles externalize resources in simple text files as key/value pairs. Translators translate the shared resources and produce a locale-specific resource bundle. Java then loads the appropriate resource bundle for the current locale, displaying the localized string in the UI.</p> <p>Java also provides ListResourceBundles that allow Developers to externalize both strings and complex objects such as graphics for localization. Unlike standard Java resource bundles, ListResourceBundles are actual Java code that is compiled and bundled with the software.</p> <p>The ICU4C uses its own version of resource bundles to externalize strings for localization. ICU4C resource bundles are written in a proprietary text file format. The resource bundles are compiled into .res files that are distributed with the software release.</p>

### 3.2 Purpose

This document is intended to outline the workflow and responsibilities for localizing a release of the Globus Toolkit. The intended audience for this document are Globus Developers, Globus community members, or anyone else who may wish to localize the Globus Toolkit for a particular locale.

Globalization of the Globus Toolkit will produce US English resource bundles, and the Globalization effort will include production of localized resources for a second Locale to ensure the Globalized Toolkit functions properly for a non English Locale.

The Implementation Localization packages for additional Locales may be done either by the Globus Alliance or by others. It is assumed that Community members with interests in specific Locales may be willing to implement language packs for those Locales and contribute them to the Globus Toolkit.



## **4. OVERVIEW OF GLOBUS TOOLKIT L10N**

This section provides an overview of the Globus Toolkit L10N process. For detailed guidelines on each of the steps in the L10N process, see Section 8, Localization Guidelines.

There are two steps to produce a Locale specific version of the Globus Toolkit: the first is Globalization of the Toolkit to prepare it for Localization. This is described in the Globus Toolkit Internationalization Coding Guidelines document. The procedures described below assume that the Toolkit has been Globalized as described in the Globus Toolkit Internationalization Coding Guidelines and has been reviewed for I18N Compliance.

Localization of the Toolkit for particular Locales may be done by members of the Globus Alliance with full commit rights to the repository. But Localization for specific Locales could be done by Community Members without Globus Toolkit Repository commit rights. The localization procedures are the same in both these cases, but Globus Contributors will submit their localized resources as a contribution to the toolkit, and Globus Developers will merge the contributed resources into the Globus CVS Repository. Globus Developers will commit localized resources directly to the Globus CVS Repository. This workflow will be revisited when Globus Toolkit source control moves to a more open model that enables contributors to submit code to selected modules in the source repository. The workflow is described below. See the figure on the next page for a diagrammatic view.

Globus Contributors developing a Localization pack will capture a snapshot of the Globus Toolkit CVS Repository while Globus Alliance Developers would create branch in the Repository. To ensure that the localized resource bundles include resources added during Globus development, Globus Contributors working from a snapshot will need to regularly merge changes from the Globus CVS Repository into their snapshot.

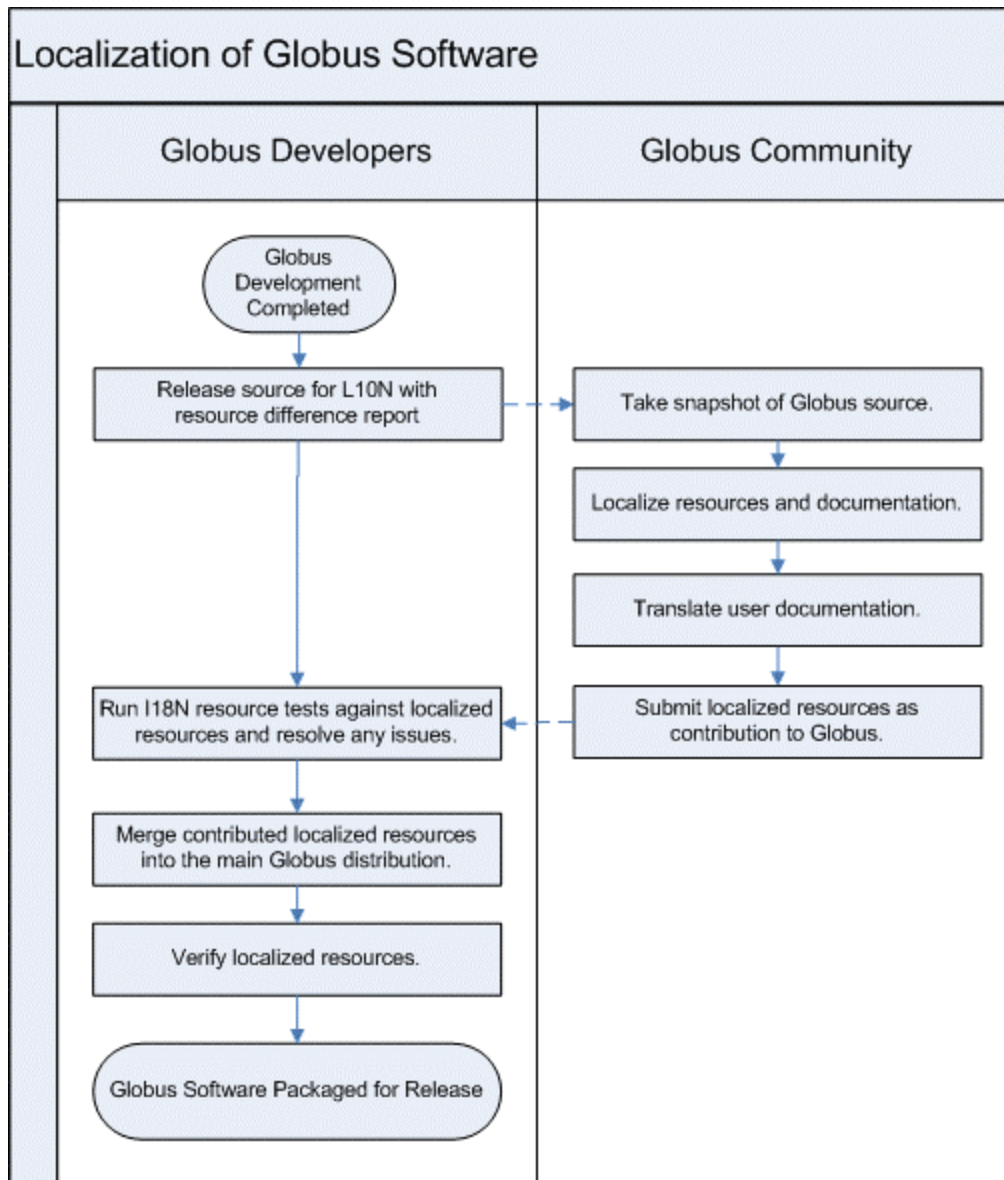
First, the localization team will prepare empty resource bundles for the locale. Then translators fill in the resource bundles for the new locale and translate any new or updated resources in the resource trees. These new resource bundles will be included in the Globus Toolkit distribution.

Resources that shall be localized include:

- Java ListResourceBundles
- ICU Resource Bundles for C code
- XML resource files
- Globus Toolkit script resource files
- Installation resource files
- Globus Toolkit user documentation

Figure 1: Globus Toolkit Localization Process provides an overview of the Globus Toolkit L10N process. Once development has been completed on a Globus Toolkit release, the Globus Developers complete the following steps to prepare Globus Toolkit resources for L10N and incorporate localized resources into the Globus Toolkit release:

**Figure 1: Globus Toolkit Localization Process**



- The Globus Developers release the Globus Toolkit revision for localization, providing the resource difference reports generated by the Globus build.
- The L10N developers check out a snapshot of the Globus Toolkit code to be used in L10N.
- The L10N developers translate any new and updated resources. The L10N developers also translate the current user documentation for tested locales.
- Once translation is complete, the L10N developers submit the localized resources as a contribution to Globus Toolkit.
- When the L10N resources are available, Globus Developers merge the localized resources into the Globus Toolkit repository and run the localized resource build that includes the I18N resource tests. These tests verify that all resources included in the

primary resource tree are resolved in the resource trees for all tested locales and that all resources changed for the Globus Toolkit release in the primary resource tree have changed in the resource trees for all tested locales.

- The Globus Developers contact the community members who provided the localized resources to resolve any errors found in the resource tests and rerun the tests with corrected resources.
- Once the translated resource bundles have passed the I18N resource tests, the Globus Developers merge the contributed localized resources into the mainline, update the installation to include the new resources, and rerun the installation tests to confirm that the localized resources install properly.
- Once all I18N tests have been passed, Globus Developers package the Globus Toolkit release including the localized resources.

For detailed guidelines on each of these steps in the L10N process, see Section 8, Localization Guidelines.

## **5. HARDWARE AND SOFTWARE REQUIREMENTS FOR L10N AND TESTING**

This section covers hardware and software requirements for the Globus Developers and localization Globus Contributors to meet the requirements for G11N on a Globus Toolkit release. In many cases, setup of a international development and testing environments can be costly and labor intensive, so these requirements should be reviewed early in a release and ample time allowed to acquire, install, and configure the required hardware and software.

### **5.1 Localization Support Files to Be Distributed with the Globus Toolkit**

L10N will produce a collection of resource bundles and other resource files that provide translated strings and other localized resources for each tested locale. In addition, some locales may require additional fonts and other support files to ensure the proper display of localized content. In most cases we will rely on the localized version of the host operating system to provide the necessary fonts and supporting applications to properly display localized content. However, we may elect to provide other support files for a locale when Globus Developers determine that explicitly providing a font or other supporting file will minimize risk and testing time for a given locale. Should we elect to distribute supporting files, these files must be included in the installation and verified by installation testing as described in 8 Localization Guidelines.

### **5.2 Hardware and Software Requirements for Globus Developers**

To properly implement and test I18N on Globus Toolkit code, the Globus Developers will need the following additional resources:

- Developers will need development and unit test environments with language support for appropriate target locales. This will typically involve installing multiple language packs on both Linux and Windows, so Globus Developers should allow additional time for setting up their development environments.

Note that language packs can take up a lot of space and non-ASC-II processing can be resource intensive. Globus Developers will need more storage and possibly additional memory on their development stations to accommodate multi-lingual support. To minimize resource requirements, we may want to set up a dedicated development station for development and testing of G11N features.

- For testing we will need test machines on each of the target platforms with language support for all tested locales. Note that we do not need a separate machine for each target locale. In most cases, we can use a single test machine per platform that has language support for all tested locales installed, and we can switch between locales during testing. These test machines will need a full Globus Toolkit installation and must meet the

hardware and software requirements for a Globus Toolkit workstation. Note that language support can take up a lot of space, so I18N test PCs may need additional storage to accommodate multi-lingual support.

- Globus Developers will need a means of inputting non-ASC-II text for test purposes. The best approach is to cut and paste valid non-ASC-II text from a prototype file. Alternately, non-ASC-II text can be input using Input Management Editor (IME) software that allows users to enter Unicode data without special hardware, but it is better to copy and paste content generated by translators or translation programs.
- Localized versions of various supporting applications such as browsers and text editors. In many cases, this will simply involve installing additional language support for a software package.
- We will need to build a library of localized test content from all tested locales. This will include job configurations, supporting files, and other localized input, and each input file should include proper output files for automated comparison during testing. Since we need valid localized data, we may need to contract a L10N resource to assist with the production of test data.

### **5.3 Hardware and Software Requirements for Localization**

In addition to their preferred suite of translation tools the contributors performing L10N will require:

- A Globus test station to bench test localized resources. This could be a VM Ware image. The community members doing L10N should at least test on Linux and possibly Windows as well. The test environment will need language and font support for all target locales, and should include an input management editor (IME) for entering localized content.
- A J2SE installation for compiling ListResourceBundles.
- An ICU installation for compiling ICU resource bundles.

## **6. L10N RESOURCES**

Actual L10N of Globus for various locales may be conducted by various interested parties as funding is available. However, L10N is may be important to the acceptance of the Globus Toolkit in the target locale. Use the following guidelines when selecting L10N resources for a Globus Toolkit release:

- Translators for a target locale should be fluent in the target language and familiar with the locale. Native speakers are preferred.
- Translators should be familiar with software translation, including Java and C resources files. Experience with ICU resource files is a must.
- Translators should bench test their localized content before submitting the localized resource bundles. The L10N developers must be technically sophisticated enough that they are comfortable running simple Globus jobs to test localized resource bundles.
- Select translators familiar with Linux and Windows.
- Select translators familiar with font and language support for the target locales on your target platforms. The L10N developers should be able to identify potential issues displaying localized content and recommend support fonts or other files that should be bundled with the Globus Toolkit distribution.

## 7. TESTED LOCALES

To maximize the utility of the Globus Toolkit to the international grid community, Globus Toolkit code is internationalized and supports L10N for all user locales tested by the host operating systems.

Presently, the Globus Toolkit has been localized for the following locales:

[This table is provided as a sample. When we implement localization, this table will be updated with the tested locales.]

Language	Region	Font	Notes
EN	US		
EN	UK		

## **8. LOCALIZATION GUIDELINES**

This section provides detailed instructions for localizing a release of the Globus Toolkit, including:

### **8.1 Localizing Globus Toolkit Resources**

#### **8.1.1 *Source Management During Localization***

The community members localizing Globus Toolkit for a particular locale take a snapshot of the Globus Toolkit distribution at a particular release and perform L10N on a local copy of the Globus Toolkit source. It is recommended that the contributors maintain an additional source control system to save incremental changes to localized resources.

Once L10N is complete, the community members submit the localized resources as a contribution to the Globus Toolkit. When time allows, Globus Developers test the contributed localized resources and merge them into the Globus Toolkit distribution.

#### **8.1.2 *Installing Globus Toolkit and Testing Translated Resources***

The Globus Developers will test localized resources, but the community members performing localization should have a Globus Toolkit installation for each tested locale available to test and debug localized resources during translation.

#### **8.1.3 *Localizing the Java ListResourceBundles***

Java code in the Globus Toolkit saves localized content in ListResourceBundles. For each Java component stores resources in a ListResourceBundle named Resource.java, and the appropriate locale string shall be added to the file name and class name for localized versions of the ListResourceBundle. JavaListResource bundles for each Globus component will reside in the org.globus.<componentName>.util package.

Each resource bundle is assigned a name, and localized versions of the resource bundle use the same name with the locale string appended to the end of the name. For example, the wsrf package in the Globus Toolkit currently uses the org.globus.wsrfultils.Resources resource bundle saved as Resources.java. Localized versions of the resource bundle might include:

- Resources\_de.java
- Resources\_de\_DE.java
- Resources\_en.java
- Resources\_en\_UK.java

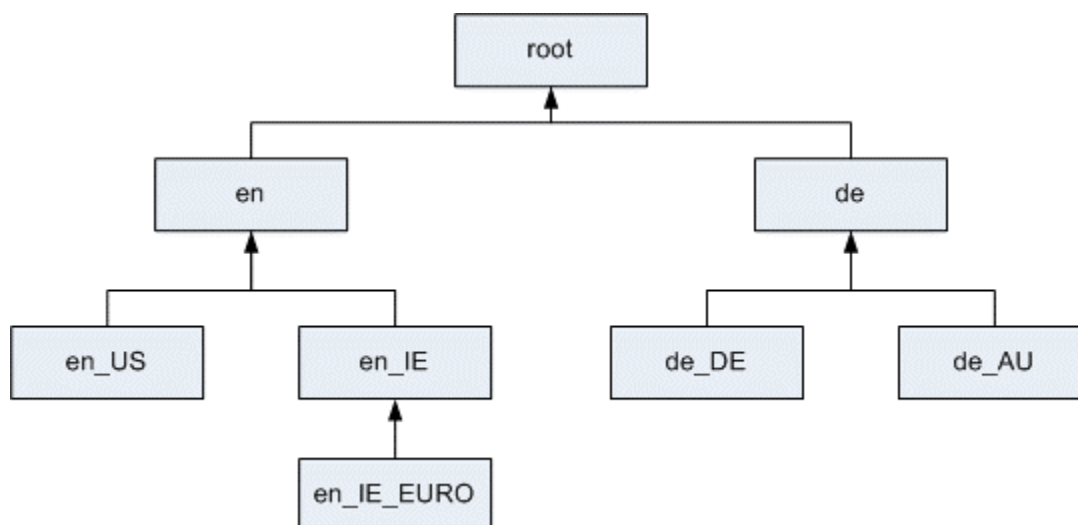
In Java, the locale string appended to the resource bundle name is composed of three parts:

- Language: The two letter, lower-case, ISO-639 language code for the language. See <http://www.unicode.org/unicode/onlinedat/languages.html> for a list of language codes.

- Region: The upper-case ISO-3166 country code for the region. See <http://www.unicode.org/unicode/onlinedat/countries.html> for a list of valid country codes.
- Variant: A special code used to specify finer grain locale settings. Java allows you to specify multiple variant settings separated by an underscore. For example, WIN\_EURO would denote a Windows system in a country that accepts Euros, and LINUX\_EURO would denote a Linux box in a country that accepts Euros. The EURO variant code is a Java standard variant code used to specify countries that accept Euros.

Resource bundles are managed in a tree structure ranging from the default resource bundle to the most specific resource bundle. For example, the Figure 2: Sample Resource Bundle Tree shows a collection of resource bundles for a system where the default locale is US English (en\_US). Note that we have a more general resource bundle for English (en). Beneath that we have resource bundles for US, UK, and Ireland. Beneath Ireland's resource bundle, we have a resource bundle for Ireland English with Euro support. On another branch of the tree, beneath the German (de) resource bundle, we have more specific resource bundles for Germany and Austria.

**Figure 2: Sample Resource Bundle Tree**



When applications need a localized resource, they load the resource from the resource bundles based on the locale specified in user settings or by the application.

Based on the selected locale, Java tried to locate the localized resource in the most appropriate resource bundle, starting with the most specific resource bundle. For example, if we try to find a resource in the English/Ireland/EURO locale, Java would search for resource bundles in the following order:

- MyPackage.MyBundle\_de\_DE\_EURO
- MyPackage.MyBundle\_de\_DE
- MyPackage.MyBundle\_de
- MyPackage.MyBundle

Following this model, the L10N developers should create a tree of resource bundles for each group of related target locales, starting with a language resource bundle (en), then a language/region resource bundles (en\_US, en\_IE), and finally a language/region/variant resource bundle (en\_IE\_EURO). Note that resource bundles for different locales may share the same parent resource bundle. For example, resource bundles for en\_US and en\_IE both share the parent resource bundle for en. To minimize duplication, resources that are common across a language should be included in the most general resource bundle, so resources strings that are same for all German speaking locales should be included in the de resource bundle.

Globus Toolkit Java code externalizes user interface resources for L10N using ListResourceBundles.

List Resource Bundles are defined as Java classes. The following example shows a sample ListResourceBundle [DEIT-2001]:

```
/* When translating, add _language_country_variant to the class name.
*/

public class SampleResourceBundle extends ListResourceBundle {

    public Object [][] getContents() {
        return contents;
    }

    static final Object [][] contents = {
        /* Start translating resources here */
        {"okButton", new Button("OK")},
        {"negativeInteger", new Integer(-1)},
        {"textString", "Thank you for reading our book"}
        /* End translation here */
    };
}
```

To localize ListResourceBundles, the L10N developers do the following:

- For new locales, the L10N developers create new resource bundles for the locale and copy the resources to be translated to the appropriate ListResourceBundles for the locale.
- The L10N contributors translate all new resources and any updated resources that have been identified in the resource change list. In the sample code above, the object array that needs to be translated is marked with comments.
- Once all of the resource bundles on a given branch of the resource tree are complete, the L10N developers review the lower level resource bundles to see if there are any common resources that should be moved to a higher resource bundle. For example, after finishing German translation, the L10N developers would review the de\_DE and de\_AU resource bundles to see if there are any identical resources that should be moved to the de resource bundle.

- Once translation of a ListResourceBundle is complete, the L10N compiles the resource bundle resolves any compiler errors by calling :

```
javac package.ResourceName_lang_country_variant.java
```

Compiler errors are usually the result of an error made while translating the bundle, and the error should give a line number in the ListResourceBundle that is the likely cause of the problem.

The L10N developers should then install the localized resource on their Globus test station and do a spot check that translated resources display properly for the target locale.

- Once testing is complete, the source files for the localized resource bundles should be packaged and submitted as a contribution to the Globus Toolkit.
- The Globus Toolkit development team will run further tests on the localized resources. Globus Developers may contact the L10N contributors for assistance resolving any issues identified in testing.

### **8.1.4 Localizing the ICU Resource Files**

The ICU4C C and C++ enables Globus Developers to externalize resources that need to be localized in Resource Bundles. ICU Resource Bundles are created in a special text file format, and completed resource bundles are compiled into .res files using genrb.

The Globus Developers create completed Resource Bundles for the default locale. Resource files for a specific locale have the same name as the primary resource bundle with a locale string added. For example, the root resource file for Grid FTP would be gridftp.txt, and some locale specific resource bundles might include:

- gridftp\_en\_UK.txt
- gridftp\_es.txt
- gridftp\_es\_ES.txt
- gridftp\_es\_MX.txt

The ICU locale string is formatted:

```
language_[script_]COUNTRY_variant@ keyword=value
```

with the following components:

- Language: The two or three letter, lower-case, ISO-639 language code for the language. For lists of language codes, see <http://www.loc.gov/standards/iso639-2/> .
- Script Code: An optional script code can follow the language code. Script codes must be valid Unicode ISO 15924 script codes. For a list of script codes, see <http://www.unicode.org/iso15924/iso15924-codes.html> .
- Country Code: The upper-case ISO-3166 country code for the region. For a list of valid country codes, see <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html> . You can call the static Locale.getISOLanguages() method to obtain an array of ISO country codes supported by the JVM.

- Variant Code: A special code used to specify additional locale settings. ICU allows you to specify multiple variant settings separated by an underscore. For example, `_EURO_WIN` would denote a Windows system in a country that accepts Euros, and `_EURO_LINUX` would denote a Linux box in a country that accepts Euros. The EURO variant code is an ICU standard variant code used to specify countries that accept Euros.
- Keywords: Finally you can specify keywords with their value for the Locale. Currently, the ICU accepts the following keywords
  - Calendar: Specifies the type of calendar created by the ICU Calendar class. Possible values include “gregorian,” “arabic,” “chinese,” “civil-arabic,” “hebrew,” “japanese,” or “thai-buddhist.” For a complete list of possible values, see <http://www.unicode.org/reports/tr35/>.
  - Collation: Specifies how the collation class searches locale data when creating a Collator. For a list of possible values, see <http://www.unicode.org/reports/tr35/>.
  - Currency: A standard three-letter currency code for the region. For a list of currency codes, see [http://www-950.ibm.com/software/globalization/icu/demo/locales?\\_en&SHOWCurrencies=1](http://www-950.ibm.com/software/globalization/icu/demo/locales?_en&SHOWCurrencies=1).

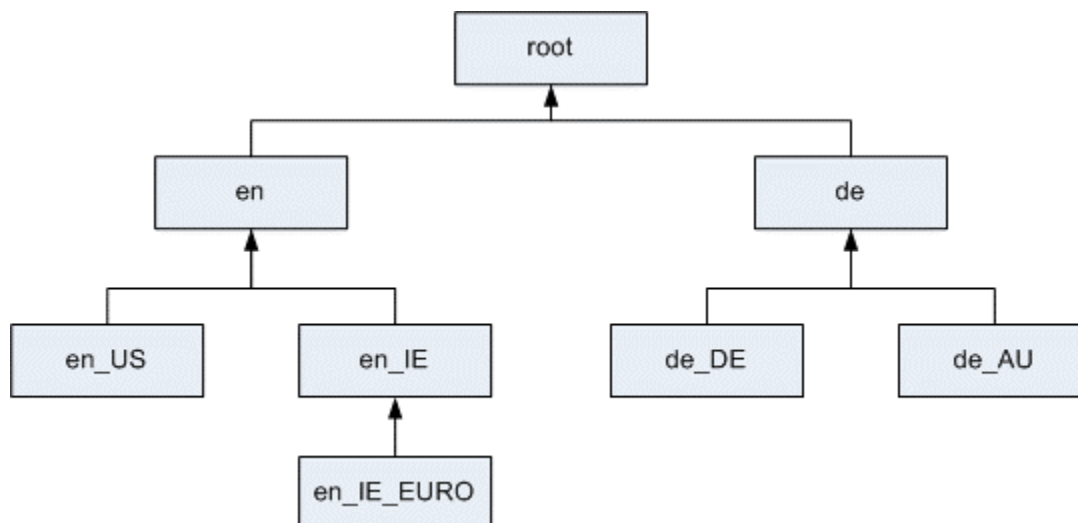
Here are a few sample ICU Locale strings:

```
En_US  
En_IE@currency=IEP  
fr@collation=phonebook;calendar=islamic-civil  
sr_Latn_YU_REVISSED@currency=USD
```

Note that a Locale string does not need to include all elements. The language, script, country, and variant codes are separated by underscores. Any key words appear at the end of the string following an @ sign. Multiple key/value pairs are separated by semi-colons. The ICU also includes methods for converting locale strings that do not follow this format to ICU formatted locales strings. These methods are primarily used for converting POSIX or .NET locals strings that may substitute a – for the \_ or use an @ sign before the variant. For more examples of Locale strings and the methods to convert non- standard locale strings, see <http://icu.sourceforge.net/userguide/locale.html>.

Resource bundles are managed in a tree structure ranging from the default resource bundle to the most specific resource bundle. For example, the Figure 2: Sample Resource Bundle Tree shows a collection of resource bundles for a system where the default locale is US English (`en_US`). Note that we have a more general resource bundle for English (`en`). Beneath that we have resource bundles for US, UK, and Ireland. Beneath Ireland’s resource bundle, we have a resource bundle for Ireland English with Euro support. On another branch of the tree, beneath the German (`de`) resource bundle, we have more specific resource bundles for Germany and Austria.

**Figure 3: Sample Resource Bundle Tree**



The ICU always tries to use the most specific resource bundle for the current locale, and the ICU falls back to more general resource bundles when a specific resource bundle is unavailable or the resource the ICU is searching for is not in the specific resource bundle. For example, suppose the current locale is `en_IE_EURO`, and the ICU is looking for the resource `TUESDAY`. To save space the resource `TUESDAY` is actually encoded in the English `en` resource bundle. The ICU would first check the `en_IE_EURO` then the `en_IE` resource bundle. When it doesn't find `TUESDAY` in either bundle, the ICU would fall back to the English (`en`) resource bundle where the `TUESDAY` resource is located. If `TUESDAY` weren't found in the English (`en`) resource bundle, the ICU would fall back to the default resource bundle for the system.

Note that the fall back mechanism can have unexpected side effects. For example, if a German user requests a resource on a Japanese server, and the resource is not included in any of the German resource bundles, the system would fall back to the default Japanese resource bundle and return Japanese text to the German user. For this reason, you must always ensure you have provided appropriate localized resources for all tested locales.

Localized resource bundles can take up a significant amount of storage space. To minimize required storage, you should take advantage of the tree structure for resource bundles, and only localize resources once in the most general resource bundle that is appropriate. For example, most German resources could be localized in the German (`de`) resource bundle. Only resources that vary between Germany and Austria would be included in the Germany (`de_DE`) and Austrian (`de_AU`) resource bundles.

Also, you should include a resource bundle for all tested locales even if there are no specific localized resources for that locale. If all the localized resources for Germany are the same as the German (`de`) resource bundle, you should still include an empty resource bundles for Germany (`de_DE`) to show that this is a tested locale.

Resource bundles can contain a combination of simple and complex data. Simple data includes strings, binary, integers, and integer arrays. Complex data is stored in either a key/value table or an array and can include data of any type.

ICU resource bundles are created in specially formatted text files and compiled into binary resource bundles using the genrb utility. To support Unicode character data, always save resource bundle source files with UTF-8 or UTF-16 encoding. Resource bundle source files can be saved with a .txt extension.

For complete guidelines on formatting resource bundle source files, see [http://dev.icu-project.org/cgi-bin/viewcvs.cgi/\\*checkout\\*/icuhtml/design/bnf\\_rb.txt](http://dev.icu-project.org/cgi-bin/viewcvs.cgi/*checkout*/icuhtml/design/bnf_rb.txt).

The following example shows a sample resource bundle source file:

```
// Comments start with a '//' and extend to the end of the line
// first, a locale name for the bundle is defined. The whole bundle is
// a table
// every resource, including the whole bundle has its name.
// The name consists of invariant characters, digits and following
// symbols: -, _
root {
    menu {
        id { "mainmenu" }
        items {
            {
                id { "file" }
                name { "&File" }
                items {
                    {
                        id { "open" }
                        name { "&Open" }
                    }
                    {
                        id { "save" }
                        name { "&Save" }
                    }
                    {
                        id { "exit" }
                        name { "&Exit" }
                    }
                }
            }
        }
    }
    {
        id { "edit" }
        name { "&Edit" }
        items {
            {
                id { "copy" }
                name { "&Copy" }
            }
            {
                id { "cut" }
                name { "&Cut" }
            }
            {
                id { "paste" }
                name { "&Paste" }
            }
        }
    }
}
```

```
        ...
    }
}

// This resource is an array, thus accessible only through
iteration and indices...
errors {
    "Invalid Command",
    "Bad Value",

    // Add more strings here...

    "Read the Manual"
}

splash:import { "splash_root.gif" } // This is a binary imported
file

pgpkey:bin { alb2c3d4e5f67890 } // a binary value

versionInfo { // a table
    major:int { 1 } // of integers
    minor:int { 4 }
    patch:int { 7 }
}

buttonSize:intvector { 10, 20, 10, 20 } // an array of 32-bit
integers

// will pick up data from zoneStrings resource in en bundle in the
ICU package
simpleAlias:alias { "/ICUDATA/en/zoneStrings" }

// will pick up data from CollationElements resource in en bundle
// in the ICU package
CollationElements:alias { "/ICUDATA/en" }
}
```

Globus Developers will provide the following resource bundles for each application:

- A root resource bundle that contains all the resources for applications with values in US English.
- Empty resource bundles for English (en) and US English (en\_US) that do not need to be translated.

The community members performing L10N should complete the following steps to localize the resource files for all tested locales:

- Create resource bundles for the target locale(s). For existing applications and locales, the translators will update existing resource bundles.
- For new locales, the L10N developers copy the resources to be translated from the root Resource Bundle to the appropriate Resource Bundle for the locale.

- The L10N developers translate all new resources and any updated resources that have been identified in the resource change list. Use the following guidelines when translating resources in ICU Resource Bundles:
  - Always save your Resource Bundle source files with UTF-8 encoding.
  - Resources in the Resource Bundles are stored in the following format:

```
name:type { value }
```

The name is the label used to identify the resource.

The type may or may not be present and specifies the data type for the resource. Some sample types include string (:string), integers (:int), and binary files (:bin).

You do not need to translate the name or data type listed in the resources file.

The value will vary by data type. The most common type will be strings that need to be translated for the locale. Numeric types may or may not need to be updated for the locale.

In the case of binary files, such as graphics, make a copy of the file for translation and update the file name in the Resource Bundle for the locale to refer to the translated file. When naming translated graphics, use the same file name as the original graphic and append the locale string to the file name on the translated version.
  - For additional guidelines on working with Resource Bundle formats, see <http://icu.sourceforge.net/userguide/ResourceManagement.html> and [http://dev.icu-project.org/cgi-bin/viewcvs.cgi/\\*checkout\\*/icuhtml/design/bnf\\_rb.txt](http://dev.icu-project.org/cgi-bin/viewcvs.cgi/*checkout*/icuhtml/design/bnf_rb.txt).
- Once all of the resource bundles on a given branch of the resource tree are complete, the L10N developers review the lower level resource bundles to see if there are any common resources that should be moved to a higher resource bundle. For example, after finishing German translation, the L10N developers would review the de\_DE and de\_AU resource bundles to see if there are any identical resources that should be moved to the de resource bundle.
- Once translation of a branch of the Resource Bundle tree is complete, the L10N developers compile the resource bundles resolves any compiler errors by calling:

```
genrb -package-name GLOBUSRES [list of input files]
```

Compiler errors are usually the result of an error made while translating the bundle, and the error should give a line number in the Resource Bundle source file that is the likely cause of the problem.
- The L10N developers should then install the localized .res files on their Globus test station and do a spot check that translated resources display properly for the target locale.
- Once testing is complete, the source files for the localized resource bundles should be packaged for submission as a contribution to the Globus Toolkit.
- Globus Developers will run further tests on the localized resources before the contributed resources are merged into the main Globus Toolkit distribution. Globus Developers may

contact the L10N contributors for assistance resolving problems found while testing the localized resource bundles.

### ***8.1.5 Localizing Script Resource Files***

Script resource files shall be distributed with the Globus Toolkit release, so strings included in Perl and Shell UI Scripts can be translated. Use the following guidelines when localizing Globus Toolkit scripts:

- Retrieve the default script resource files any existing resource files for your target locale.
- Retrieve the resource change list for the Globus Toolkit release and use this as a task list for your translation.
- Save all resource files with UTF-8 encoding.
- Localized versions of script resource files should use the same file name as the resource file for the default locale with a locale string appended before the file extension.
- After localized resources have been finished and tested, submit the resources back to the Globus Toolkit as a contribution.

### ***8.1.6 Localizing Globus Toolkit User Documentation***

Use the following guidelines when localizing Globus Toolkit documentation:

- Retrieve the latest documentation for the default locale and the target locale that is distributed with Globus Toolkit release.
- Retrieve the documentation change list distributed with the Globus Toolkit release and use this report as a task list for localizing documentation.
- Save all files in UTF-8 format.
- When creating new documentation source files, use the same file name used for the default locale and append a locale string before the file extension.
- Save all files for your target locale to the subdirectory named with your locale string.
- Contribute your localized documentation back to the Globus Toolkit documentation project.

## **8.2 Verifying and Packaging Localized Resources**

### ***8.2.1 Running the I18N Resource Tests***

Before merging the localized resources into the main distribution, Globus Developers will run the following tests on the localized resource bundles:

- Verify that all resource bundles compile without error.
- Run automated tests that loop through the primary resource trees for Java and C and verify that all resources referenced in the primary resource tree resolve in each of the other locale specific resource trees and that all resources changed in the primary resource tree have changed in the resource trees for other locales.
- Verify that all resources referenced in the XML and script resource files for the primary locale are also referenced in all XML and script resource files for all other locales.

Globus Developers will work with the L10N contributors to resolve any errors identified in the resource tests. Once all errors have been resolved, Globus Developers will rerun the I18N resource tests, and resources will not be merged until all tests pass.

### ***8.2.2 Merging Localized Resources into the Mainline***

Once localized resources have passes the I18N resource tests, the localized resources should be merged into the mainline. Once the localized resources have been merged, a tagged build that includes the localized resources should be produced and submitted for testing. Builds will be tagged following Globus Toolkit conventions for build labeling and release.

### ***8.2.3 Formal Validation of Localized Resources***

Once the localized resources have been merged into the mainline and a final build has been tagged, Globus Developers will run the following I18N tests:

- A formal rerun of the I18N unit test to validate that all resources are resolved in the primary resource tree. This test verifies the integrity of the primary resource tree and that no additional resources have been referenced in code but not added to the resource bundles.
- Sanity tests to verify the following functionality on appropriate locales and platforms:
  - Display of bidirectional text.
  - Display of non-ASC-II text in the shell, log files, and other output.
  - Processing of non-ASC-II file and shell input.
  - I18N testing verifies that the Globus Toolkit can properly process and display non-ASC-II, bidirectional, and other localized content.
- Installation tests to verify that:
  - Resource files install properly on all platforms for new installation and overinstalls.
  - Any additional fonts and supporting files install properly on the appropriate platforms and locales.

The validation tests will be developed as part of the G11N implementation project, and developing these tests is outside the scope of the current G11N evaluation effort.

#### ***8.2.4 Packaging Localized Content for Release***

Once all tests have passed, the localized resource bundles shall be packaged with the Globus Toolkit release.

## 9. SUPPORT

See the following resources for additional information on I18N information:

- For more information on using the ICU4C and ICU4J, see:
  - <http://icu.sourceforge.net/userguide/>
  - <http://icu.sourceforge.net/apiref/>
  - <http://developers.sun.com/dev/gadc/educationtutorial/creference/index.html>
  - <http://icu.sourceforge.net/docs/>
- For more information on Java I18N, see:
  - <http://java.sun.com/j2se/corejava/intl/>
  - <http://java.sun.com/docs/books/tutorial/i18n/>
  - <http://java.sun.com/j2se/1.5.0/docs/api/index.html>
  - <http://www.unicode.org/iuc/iuc15/ta4/slides.pdf>
  - <http://forum.java.sun.com/forum.jspa?forumID=16>
  - <http://www.joconner.com/javai18n/>
- For more information on XML and Web Service I18N, see:
  - <http://www-128.ibm.com/developerworks/library/x-globe/>
  - [http://www-128.ibm.com/developerworks/websphere/techjournal/0409\\_tong/0409\\_tong.html](http://www-128.ibm.com/developerworks/websphere/techjournal/0409_tong/0409_tong.html)
  - <http://www.w3.org/International/ws/>
- For standard country codes, see <http://www.unicode.org/unicode/onlinedat/countries.html>.
- For standard language codes, see <http://www.unicode.org/unicode/onlinedat/languages.html>
- For general Unicode information, see <http://www.unicode.org/>

For specific questions on Globus Toolkit G11N contact:

Robert Gaffaney at [gaffaney@univa.com](mailto:gaffaney@univa.com) or Patrick Doran at [doran@univa.com](mailto:doran@univa.com)