



# Software Internationalization

---

WHITE PAPER

**Author: Shridhar Rajgopalan**

---

As more and more software applications are developed for user community across the world, Internationalization of the software is a key concern. From a mere, application design consideration, Internationalization has become a small field by itself and significant amount of research has been carried out in this field. While there is a large number of a literature available in the industry on various techniques of Internationalization, there are few survey papers. There is a need for understanding Internationalization in a holistic perspective and this paper addresses that need.

Based on a literature survey and practical experience of implementing Internationalization, this paper tries to present Internationalization in a holistic sense. It covers various aspects like need for Internationalization, sub areas of Internationalization, typical problems involved in implementing Internationalization, design guidelines and some of the recommended solutions.

## Table of Contents

<b>INTRODUCTION .....</b>	<b>3</b>
<b>CHALLENGES TO BE ADDRESSED IN INTERNATIONALIZATION .....</b>	<b>4</b>
FUNCTIONAL CHALLENGES .....	4
BUSINESS ISSUES .....	4
TECHNICAL ISSUES .....	5
CULTURAL ISSUES .....	5
DOCUMENTATION ISSUES .....	5
<b>DESIGN GUIDELINES FOR AN INTERNATIONAL SOFTWARE .....</b>	<b>5</b>
GENERAL GUIDELINES .....	6
USER INTERFACE DESIGN GUIDELINES .....	6
DATABASE DESIGN GUIDELINES .....	7
DOCUMENTATION GUIDELINES .....	7
<b>INTERNATIONALIZATION SOLUTIONS .....</b>	<b>8</b>
FUNCTIONALITY SOLUTIONS .....	8
BUSINESS SOLUTIONS .....	9
TECHNICAL SOLUTIONS .....	10
<b>ISSUES POSED BY AN INTERNATIONALIZED APPLICATION .....</b>	<b>13</b>
<b>CONCLUSION .....</b>	<b>14</b>
<b>REFERENCES .....</b>	<b>14</b>
<b>ACKNOWLEDGMENT .....</b>	<b>14</b>
<b>ABOUT THE AUTHOR .....</b>	<b>14</b>
<b>ABOUT WIPRO TECHNOLOGIES .....</b>	<b>15</b>
<b>WIPRO IN E-BUSINESS .....</b>	<b>15</b>

## Introduction

As companies weave e-Commerce/e-Business on a global scale into their fundamental business processes there is a need for the software internationalization. This gives inherent advantages such as increased revenue, decreased expenses and better customer communications and savings to their prospective customers, established customers and active partners. Internationalized software results in an exponential increase in customer satisfaction, which can increase sales in customer support communications; in enhanced global information dissemination and in a better return on IT investments.

Software internationalization is the development process using libraries that enable one single application to work with text in any language, for any place in the world. For example, instead of having software versions for ten different countries, you can use a framework to create one version that can work seamlessly and transparently in many different and unique countries. Internationalization is also commonly referred as I18N('I18N' is a standard abbreviation for "internationalization", which starts with the letter I, ends with the letter N, and contains a total of 18 letters).

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. An internationalized program has the following characteristics:

- With the addition of localized data, the same executable can run worldwide
- Textual elements, such as status messages and the GUI component labels are not hard-coded in the program. Instead they are stored outside the source code and retrieved dynamically
- Support for new languages does not require recompilation
- Culturally dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language
- It can be localized quickly

Localization (L10N) is the process of adapting software for a specific region or language by adding locale-specific components and translating text. Usually, the most time-consuming portion of the localization phase is the translation of text.

An internationalized application can display information differently throughout the world. For example, the program will display different messages in Paris, Tokyo, and New York. If the localization process has been fine-tuned, the application will display different messages in New York and London to account for the differences between American and British English. Text messages are the most obvious form of data that varies with culture. However, other types of data may vary with region or language. The following list contains examples of culturally dependent data:

- |                    |                  |                                 |
|--------------------|------------------|---------------------------------|
| - Messages labels  | - Online help    | - Sounds                        |
| - Colors           | - Graphics icons | - Dates                         |
| - Times            | - Numbers        | - Currencies                    |
| - Measurements     | - Phone numbers  | - Honorific and personal titles |
| - Postal addresses | - Page layouts   |                                 |

There are certain challenges, which should be addressed by software Internationalization. Some of these challenges are outlined in the next section.

## Challenges to be addressed in Internationalization

### Functional challenges

Date time handling for centralized deployment

Application software having single installation for multiple geographies needs to maintain the time context for the user. The application needs to handle various user timezones as well as the daylight saving hours, where applicable.

### Master data support

An application can have master data for various entities. The description for the master data should be supported in various languages based on the end user preferences. For example, for an order processing system, the different settlement methods should be displayed in user's preferred language.

### Integrated subsystems

Software systems sometimes consist of subsystems that to a certain extent are independent of the main system. This independence can lead to a lack of information about the system and the subsystem or, more precisely, it can lead to lack of coordination. For example, a word processor can use a third party developed spell checker which is not internationalized.

### Home page

Presenting the home page to the users in an internationalized web application is a challenge in itself.

### Notification

Most of the internationalized software sends notification to the concerned parties based on certain events. For example, in a Forex Trading System, on creation of an order, the user is informed about the details of the order and the beneficiary is informed on receipt of the payment. The system should have an in-built capability to notify information in user's preferred language.

### Business issues

Internationalized software should be aware of the regional business issues. These issues can be broadly classified as

- a) Rule driven
- b) Parameter driven

#### Rule driven

Rule driven issues are the ones where the rules governing the application logic would vary from one place to other. For example, an internationalized tax calculator should be aware of the tax rules prevalent in various countries. Similarly any software having an in-built pricing module should be aware of rules for pricing computation for varying geographies.

#### Parameter driven

There are certain business features, which can be parameterized as their values vary across multiple geographies. For example, in a foreign exchange trading system, interest earned on the security deposit for a forward contract would be 0.8% in one region and 1.2% in some other region.

Some of them can be a combination of rule and parameter driven.

## Technical issues

Some of the technical challenges for internationalization are:

- Handling of status messages, error messages, and GUI component labels
- Handling of compound messages in different languages
- Number formatting to address decimal point, thousand-separator etc.
- Currency formatting
- Date and time format
- String comparison
- Locale sensitive string operation

## Database

Database support to internationalized characters.

## Cultural issues

Some of the cultural issues, which need to be addressed in internationalized software, are:

- Icons
- Use of left arrow key
- Sound related
- Advertisements etc.

## Documentation issues

No internationalized software would be ready for distribution unless it has supporting user-required material in user's language. Some of these are:

- Help screen
- Manuals
- Other supporting documents

These issues can be addressed by some of the solutions discussed in the later part of this paper.

*"Internationalization of the software is a thought that needs to be applied at the beginning of development of the software rather than an afterthought."*

To internationalize the software, as a first step, the requirements for the software need to be understood properly. Secondly, a sound, robust design for internationalized software is a must. Some of the design aspects are covered in the following section.

## Design Guidelines for an international software

The Design guidelines are broadly divided into the following categories:

- General
- User interface
- Database

## General guidelines

Some of the general guiding design principles are:

- Fundamental design principle for Internationalization: “*Separate what is changing from what is not*” should be followed
- Patterns that can be used for Internationalization, “Abstract factory” and “Strategy” are well suited for having different rules for different locales
- Internationalization should be taken into consideration when the software is designed for the first time. Changes after the software is developed are time consuming, costly and some times frustrating
- One should have written specification as to what needs to be done at the time of product inception or design itself
- Design the product in such a way that realization of single source code for all the locales could be achieved
- Languages and locales should be known first so that translation effort could be incorporated into the application cost initially
- Should be designed in such a way that customization effort is minimum
- Identify language dependent features and develop them as components. Example: Spell Checker and Grammar Checker in Editor Software.

## User interface design guidelines

### Icons and bitmaps

- Use different icons in different countries
- Load them at run time
- Do not use any characters in icons. For example, representation of Run by “R” should be avoided
- However if it becomes unavoidable to represent icons through a character, dynamically load the character on the button
- Keep bitmaps simple
- Avoid using any symbol representing animals, religious and national emblems
- Avoid using icons based on hand gestures and other body languages as this could mean different in different countries
- Do not represent left arrow key for “Undo” in a language editor
- Be culturally sensitive about beep sound. Error representation by beep may be liked in USA but may not be liked in Japan

### Menus and dialogs

- Length of English text might grow when translated
- Design menu bars, status bars toolbars and title bars to allow text size to increase
- Design your menu bar in such a way that it does not wrap into second line
- As a thumb rule keep 30% margin in static texts and other captions
- If possible, get the captions and static texts translated into application supported languages and verify them in User Interface before freezing them
- Load all the captions and static texts through string resource table
- If resource re-compilation is to be avoided, keep all captions into a database or property file

- Keep a provision to change the language option through a configuration file
- Keep a provision to change the font and its size for all dialogs through the style sheet
- Keep the provision of setting of sizes of critical resources through style sheet
- If possible, have a UI based tool for configuration/localization for larger products
- Never hard code the size of the message box. Auto adjustment depending upon the size of the message string should be allowed
- Keep a provision to select locale specific stylesheet runtime
- Sometimes there is a restriction on the size of a menu item. The menu item size may exceed the allocated size in some language. In such cases it is preferable to show portion of the text fitting into the allocated size. This menu item can be superimposed with a tool tip, which displays the complete text

### Database design guidelines

- In configuration or set-up of the applications, database should be created dynamically
- Script should have different column lengths for Double Byte Character Set (DBCS) and Single Byte Character Set (SBCS) languages. This is more important if the same version of the software would have multiple installations across geographies. However, for centralized deployment, appropriate character set should be chosen
- Integer and double fields do not change, provided transaction value does not change. If transaction value changes, incorporate rules in the script depending upon language
- Store date as varchar2 if not very sure of the target language and do the required processing in the application such as sorting. This solves the date format problems
- Memory allocation for the equivalent variables in the front-end code, should be based on the column lengths obtained dynamically unless very sure of the lengths in DBCS and SBCS languages
- Freezing of database design should be done first, and then application design or user interface design should be taken up
- While dealing with the different currencies, care should be exercised to allocate field size properly (particularly for currencies like Italian Lira and Japanese Yen)
- While creating the database, it should be created with proper character set

### Documentation guidelines

- Keep the audience in mind. Collect the following information for the target audience:
  - Customs
  - Beliefs
  - Learning styles
  - Sensitivities
  - Taboos
  - Values
  - Presentation styles
- Avoid uniquely region specific examples. State what something is, rather than what it is called in your part of the world
- Be aware of the cultural sensitivities
- Avoid humor. This is often a cultural issue. Items appearing funny in one region may not be readily understood in other, even worse, may appear offensive
- Use simple sentences and simple words
- Use consistent nomenclature and style

- Avoid jargon, slang and idioms
- Use visuals liberally
- Limit use of acronyms
- Write large numbers in numerals
- Use the names of months in dates

## **Internationalization solutions**

The following section discusses solutions for some of the concerns associated with Internationalization.

### **Functionality solutions**

#### **Date-time handling for centralized deployment**

More and more systems are being designed for centralized deployment. Consider a centralized system catering to users from different geographies, dealing with transactions involving time. In such cases, the system should have a repository of different time-zones belonging to different sets of users with offsets from the base timezone. Additionally, any registered user with the system should have an associated timezone.

When the user performs the transaction and sees the transaction history, the user should see his/her local time for the transaction. The data repository holding the transactional history should hold user's locale timestamp for the transaction.

Consider a case when the user performs the transaction and enters the date time for future follow up to be performed by the system. In such cases, the transaction should have the user timestamp and also the system timestamp. Any batch process working on such transactions should not only look at the user/system timestamp but should also consider the daylight saving hours.

#### **Master data support**

A robust internationalized application should store the master data in different languages. Each such data should have the data and the language identifier as the composite primary key for retrieval/update of the data description.

For example, for an order processing system having multiple settlement methods, each settlement method (which is a master data) should have SettlementId, LanguageId and SettlementDescriptor associated with it. While displaying the choice of settlement methods to the user, different SettlementDescriptors corresponding to the user LanguageId should be fetched and displayed.

The application internally can work with the SettlementId (so that the application is language independent).

#### **Integrated subsystems**

Additional care should be taken to integrate third party developed components/subsystem into the application. The application owner needs to ensure that these components/subsystems provide the same internationalized support as the main application.

### Site navigation

One typical issue for internationalized software is to decide how to present the home page of the Website to the user.

Some of the solutions are:

**Alternative 1:** The homepage is shown in most frequently used language to the end user, say English. However, user is also given a choice to select one of the languages, which the site supports. The choice of languages would appear in the language specific format. Once the user selects the preferred language, the home page is displayed again in the user-selected language. Further site navigation is in user selected language.

**Alternative 2:** When the user requests for the home page, the HTTP request contains the information about the user's most preferred language (taken from the user browser setting). Further pages are thrown to the user in this language.

**Alternative 3:** For the registered users of the site, the information about their preferred language is uploaded through batch process. When the user logs to the site, the user's preferred language information is picked up from the data repository and further site navigation is done in this language.

Additionally, the sites giving facility for the users to register, should pose the questions required for the user registration in user's preferred language.

### Notification

For the users of the system, to get the notification in the form of e-mail, fax files etc,

- Language dependent templates should be defined in the system
- Application component should forward the dynamic data to be filled in the template in the recipient's preferred language
- Notification subsystem should have the capability to handle this

## Business solutions

### Rule driven

Business rules defined in the system should be restricted to a component/set of components. Also, as far as possible they should be highly configurable. This kind of design would ensure that any change in the rule could be addressed by changing the configurable parameters or changing the component(s) with minimal impact to the overall system.

### Parameter driven

The 'configurable' parameters, which vary across regions and help in decision-making, should be configured in the data repository.

It is important to identify rule and parameter driven features in the system upfront.

If a software is being designed to be distributed separately to multiple regions, then components providing business solutions can be developed separately. These components can be packaged appropriately for distribution to a particular region.

For software, which is to be deployed centrally, the business components need to be generic, flexible and configurable for plugging in needs on any new region.

There are obvious advantages and disadvantages of both the approaches.

Maintaining multiple components for different regions would make software maintenance difficult. However, any new requirement/business change for a region can be handled easily as the need can be addressed by a new component, which is custom made for this requirement.

On the other hand, a well thought out generic framework to address various business needs is easily maintainable and gives time-to-market advantage when requirements for a new region needs to be plugged in. This also means that a good amount of time needs to be spent on studying and understanding the needs of different regions before finalizing the framework. An ill thought out framework could become a nightmare over a period of time.

## Technical solutions

Solutions to some of the technical concerns raised earlier are discussed here. The solutions are more specific to Java™ 2 Platform, Enterprise Edition (J2EE).

The Java™ 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multi-tier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

### Some of the solutions are:

#### Handling of status messages, error messages, and GUI component labels.

To avoid expensive translation, isolate Translatable Text in Resource Bundles. Resource Bundles are used to isolate locale sensitive data. Resource Bundle is backed by the property file, which contains the message text with its identifier.

To handle status/error messages, an identifier uniquely identifies each of such messages. Appropriate static message against this identifier is stored in the property file. The message could also contain placeholders for the dynamic filler(s) that can be supplied by the application.

#### Handling of compound messages in different languages

Programs often need to build messages from sequences of strings, numbers and other data. If a message built from sequences of strings and numbers is hard-coded, it cannot be translated into other languages. The Java class *MessageFormat* provides a means to produce concatenated messages in language-neutral way. The *MessageFormat* object takes a set of objects, formats them, and then inserts the formatted strings into the pattern at the appropriate places.

#### Number formatting to address decimal point, thousand-separator etc.

Programs store and operate on numbers in a locale-independent way. Before displaying or printing a number, a program must convert it to a string that is in a locale-sensitive format. For example, in France the number 123456.78 should be formatted as 123 56, 78, and in Germany it should appear as 123.456,78.

Locale-specific formats for numbers, currencies, and percentages can be obtained using the factory methods provided by the Java *NumberFormat* class.

**Currency formatting**

Java `NumberFormat` class can handle currency formatting. Care should be taken to use `BigDecimal` for handling large number (say amount in Yen) and not `Double`.

**Date and time format**

Date and time formats also differ from place to place. For example, Germans recognize 20.4.98 as a valid date, but Americans expect that same date to appear as 4/20/98. The Java `DateFormat` class provides predefined formatting styles that are locale-specific and easy to use. The Java `SimpleDateFormat` class gives features to customize in locale specific formats.

**Character set conversion**

The Java platform uses Unicode as its native character encoding; however, many Java programs still need to handle text data in other encoding. Java therefore provides a set of classes that convert many standard characters encoding to and from Unicode. Java programs that need to deal with non-Unicode text data will typically convert that data into Unicode, process the data as Unicode, then convert the result back to the external character encoding. The `InputStreamReader` and `OutputStreamReader` Java Classes provide methods that can convert between other character encoding and Unicode. Also at the same time it is important to use the conversion only when required; like conversion of some binary data may be undesirable.

**Locale sensitive string operation**

Programs frequently need to manipulate strings. Common operations on strings include searching and sorting. Some tasks, such as collating strings or finding various boundaries in text, are surprisingly difficult to get right and are even more difficult when multiple languages must be considered. Java provided `Collator` class could perform this.

**Content negotiation**

When an application offers multiple language support, it needs a mechanism to select the language variant that is most appropriate for the user and the web-client. Most applications that have multiple language support provide some mechanism to select/change the language of choice. This process is called content negotiation. Implementation of content negotiation can be done either from within the application or from outside as a separate application. This is basically done in two ways:

**1. Server-driven content negotiation:**

The server selects a variant of the resource to be served to the client based on information contained in the request. This can be one or more of the request headers, the requested URL or any other available information. This process will be transparent to the end user.

**Implementation of server-driven content negotiation**

There are several HTTP request headers that help in implementing server-driven content negotiation by giving the server hints as to what the user prefers. One such header is "`Accept-Charset`".

Example: `Accept-Charset: UTF-8, ISO-8859-1, UTF-7; q=0.9.`

A client that sends this header indicates that it can display text encoded in UTF-8, ISO-8859-1 and UTF-7. The quality values (indicated by `q`) represent the client preferences. The default value is 1 that is the highest value.

Another very useful header is “*Accept-Language*”.

Example : `Accept-Language: en_US;q=1.0,en;q=0.5`.

## **2. Client-driven content negotiation:**

The method of creating screens in this way also remains same as above. In this method, the server offers a list of choices to the client so that, the user can choose the variant required. Once the choice of language is known the server can use it to continue serving pages in the same language till there is change.

The most opted technique is the mixture of both, where the server proposes initial local and the user has the option of changing it.

### **Cultural issues**

Culture can be one of the limiting factors in the internationalization process. People with different mental programming can perceive the same object in different ways. What is obvious for one community may not be obvious for other. Symbols, which in some countries represent political or religious forces, could be construed objectionable in others. The sounds coming from the clip media may not be same for all countries. For example, emergency sound. Other such examples which can be found objectionable could be advertisements, clip-art, left arrow keys etc.

This problem can be solved by having

- a) Appropriate localized version of such elements, which are loaded dynamically, based on user profile.
- b) Avoiding such elements where it is possible.

### **Java I18N/L10N toolkit**

This toolkit makes Internationalization and Localization easier by helping with some key processes:

- Checking Java files for methods, classes, class constants, and strings that are locale-specific
- Checking for hard-coded or inconsistent message strings
- Translating message strings and protecting strings that should not be translated
- Generating new resource bundles

The toolkit has four independent tools, which can be used separately. They are:

- I18N Verifier to test whether a program is international or not and suggest corrections
- Message Tool to find and optionally correct hard-coded or inconsistent messages
- Translator to translate messages in a resource bundle file into the target locale language and protect strings that should not be translated
- Resource tool to merge more than 2 resource files to a new resource bundle or find the differences in the resource files

**Jakarta Project: I18N tag library**

The I18N custom tag library from Apache contains tags that can be used in JSPs and help manage the complexity of creating internationalized web applications. Some of the tags it provides are:

Tag	Description
Bundle	Define a resource bundle for use by other I18N tags
Message	Displays internationalized text from a resource bundle
MessageArg	Specifies arguments to be used by the message tag's MessageFormat
Ifdef	Allows JSP to be evaluated (or not) on a per locale basis
Ifndef	Allows JSP to be evaluated (or not) on a per locale basis
Locale	Defines a locale context
FormatString	Formats a string
FormatNumber	Formats a number
FormatCurrency	Formats currency
FormatPercent	Formats a percentage
FormatDateTime	Formats a date/time
FormatDate	Formats a date
FomatTime	Formats a time

**Issues posed by an internationalized application**

Internationalized software also poses certain concerns. Some of them are:

- a) Locale-dependent collation is generally slower than binary comparisons. Some of the reasons are:
  - While ASCII is single-byte, UTF-8 and UTF-16 are multi-byte encoding
  - Often client data is accepted in native encoding and stored in Unicode, requiring conversion
  - Unicode transformations require more storage space than native encoding generally
- b) In a centralized deployment, a software application supporting large number of locales would be somewhat performance intensive. This problem is non-existent where there is a localized deployment and the resource bundle and other resources loaded for an application are specific to the region.
- c) Sorting in an internationalized application may involve significant overhead. In general, using the native support and yet retaining the internationalized features is a challenge.

**Areas not covered in this paper**

This paper has dwelt on the internationalization aspect of application software. This paper gives some of the suggested solution by the universally accepted programming language Java. Though other programming languages such as Perl also give internationalization support, this paper does not get into the details. Some of the other issues, which are not discussed in this paper, are:

- Operating system
- Hardware sensitivities (such as keyboard support etc)
- NLP (Natural Language Processing, Artificial Intelligence), for dynamic translation of data, using a universal language

## Conclusion

This paper discusses briefly about Internationalization, issues and some of the ways these issues can be addressed. It also emphasizes that internationalization should be one of the key founding stones for a software, supporting multiple geographies. A well thought out internationalized software could be customized easily for distribution across geographies. With the support provided by various most frequently used programming languages such as Java, the task of internationalizing the software becomes much simpler. However, any afterthought towards correcting this problem could result in revamping the software and could result in complete rewrite resulting in time, effort and money.

## References

- 1) 'Software Internationalization and Localization, An Introduction', by Emmanuel Uren, Robert Howard and Tiziana Perinotti.
- 2) <http://java.sun.com>
- 3) 'Designing Software for International Market and Double Byte Expertise', Wipro Technologies.
- 4) [www.javaworld.com](http://www.javaworld.com)
- 5) [jakarta.apache.org](http://jakarta.apache.org)

## Acknowledgment

I would like to acknowledge the efforts of K.R. Sanjiv, Nagaraja Gundappa, V.T. Balaji, Jacob Marcus and Ganesh Sethuraman who reviewed the paper and provided their valuable comments.

## About the author

Shridhar Rajgopalan is an Enterprise Architect with Wipro Technologies. He holds an M.Tech Aerospace Engineering from IIT-Madras and a degree in Civil Engineering. He has more than nine years of experience in technical consultancy, distributed system architecture and software development in object oriented environment. His interest areas are enterprise architecture, large application solution architecting and consultancy in application framework and reuse.



## About Wipro Technologies

Wipro is the first PCMM Level 5 and SEI CMMi Level 5 certified IT Services Company globally. Wipro provides comprehensive IT solutions and services (including Systems Integration, IS Outsourcing, Package Implementation, Software Application Development and Maintenance) and Research & Development Services (hardware and software design, development and implementation) to corporations globally.

Wipro's unique value proposition is further delivered through our pioneering Offshore Outsourcing Model and stringent quality processes of SEI and Six Sigma.

## Wipro in E-business

Wipro Technologies, a global IT solutions provider addresses the entire gamut of e-business requirements - from Content Management and Portal Solutions to B2B Integration, Enterprise Application Integration, Mobile Commerce and Web Security. Wipro's unique value proposition is delivered through our pioneering offshore development model and stringent quality processes of ISO-9000, SEI-CMM and Six Sigma.

<http://www.wipro.com/itservices/ecommerce/architecture.htm>

© Copyright 2003. Wipro Technologies. All rights reserved. No part of this document may be reproduced, stored in a retrieval system, transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without express written permission from Wipro Technologies. Specifications subject to change without notice. All other trademarks mentioned herein are the property of their respective owners. Specifications subject to change without notice..

### Worldwide HQ

Wipro Technologies,  
Sarjapur Road,  
Bangalore-560 035,  
India.  
Tel: +91-80-844 0011.

### U.S.A.

Wipro Technologies  
1300, Crittenden Lane,  
Mountain View, CA 94043.  
Tel: (650) 316 3555.

### U.K.

Wipro Technologies  
137 Euston Road,  
London, NW1 2 AA.  
Tel: +44 (20) 7387 0606.

### France

Wipro Technologies  
17, Square Edouard,  
VII, 75009 Paris.  
Tel: +33 (01) 5343 9058.

### Germany

Wipro Technologies  
Am Wehr 5,  
Oberliederbach,  
Frankfurt 65835.  
Tel: +49 (69) 3005 9408.

### Japan

Wipro Technologies  
# 911A, Landmark Tower,  
2-1-1 Minatomirai 2-chome,  
Nishi-ku, Yokohama 220 8109.  
Tel: +81 (04) 5650 3950.

### U.A.E.

Wipro Limited  
Office No. 124,  
Building 1, First Floor,  
Dubai Internet City,  
P.O. Box 500119, Dubai.  
Tel: +97 (14) 3913480.

[www.wipro.com](http://www.wipro.com)

eMail: [info@wipro.com](mailto:info@wipro.com)

**Wipro Technologies**

*Innovative Solutions, Quality Leadership*