IBM® Rational® Rapid Developer Globalization and Creating Multi-Locale Applications

Raul Ortega
ortegar@us.ibm.com

Table of Contents

Introduction

In order to meet the growing demand for globalized transactional Web applications, we need to explore new, highly productive environments for creating and globalizing Web products. The solution is Rational Rapid Developer®, that offers a model-driven, architected rapid application development (ARAD) approach to delivering optimized, n-tier, scalable, global applications.

The Business Opportunity: Globalizing Web-based Systems

IBM's vision for a new era in business computing is defined by highly adaptive companies that can instantly sense and respond to changing conditions and make business operations easily accessible to others.

IBM's vision for a new era in business computing is defined by highly adaptive companies that can instantly sense and respond to changing conditions and make business operations easily accessible to others. In IBM, this vision is called e-Business On Demand.

The conditions and operating environments for these companies have become increasingly global in nature. The key to survival for many of these companies lies in their ability to operate successfully in global markets. In order to exploit global markets, these companies must take into account the diversity of languages and cultures throughout the world.

Global products result in the efficient and effective deployment of applications worldwide with their ability to reach a wide range of users regardless of language or cultural preferences, local business practices or conventions. Integrating multilingual information and global business processes, end-to-end, across the company, and across borders with key partners, suppliers and customers is becoming an imperative, as is responding with speed in any language to any customer demand, market opportunity or external threat.

Consider the following:

- *By 2005, 70% of Web users will speak a primary language other than English. (GlobalReach 2003)*
- *According to IDC, 2001, companies that do nothing to globalize their Web sites project an average of 12% of their revenue will come from foreign sources in 2002 while companies that do globalize project that 30% of their revenue will come from foreign sources.*
- *Globalization markets are poised for growth once we are over the economic slump. As more global organizations compete for international audiences, customer expectations will be raised, looking for providers who "speak their language." (ABI Report, Oct 2002)*

The time to embrace the global e-Business On Demand vision is now.

The Technical Problem: Technical Challenges to Globalization

Globalization of Web applications has been slowed or deterred to some extent by the complexity of creating new global applications or transforming existing applications to meet the needs of global users.

While the opportunity may be obvious, globalization of Web applications has been slowed or deterred to some extent by the complexity of creating new global applications or transforming existing applications to meet the needs of global users.

The following challenges have slowed the globalization process within many companies:

- *Information about globalization is hard to get and can be confusing*
- *Training the entire team in the complexities of globalization is difficult and costly*
- *Implementing globalization is costly and time-consuming*
- *Translating a Web application can be cumbersome, costly, and inefficient*
- *Determining the effect of translation on page design and layout is awkward and time-consuming, because it can only be done at run-time in the current development paradigm*
- *Creating and managing text strings across languages is complex*
- *When translating large Web sites into multiple languages, managing the workflow can be overwhelming*

The Solution: IBM® Rational® Rapid Developer

IBM Rational software introduces Rational Rapid Developer—the solution to creating Web applications and providing localized versions in a highly productive environment.

Rational Rapid Developer enables a broad class of developers to rapidly build complex e-business applications that are portable to all leading deployment technologies.

By combining powerful visual modeling, RAD techniques and automated construction, Rational Rapid Developer enables a broad class of developers to rapidly build complex e-business applications that are portable to all leading deployment technologies. Rational Rapid Developer-based applications are reliable, scalable, secure and readily modifiable to accommodate changes in technology and business objectives. Leveraging mainstream development skills, you can deploy and maintain enterprise-class applications that integrate with an extensive range of legacy systems. Time-to-market and cost of ownership are greatly reduced.

After a brief introduction to the Rational Rapid Developer development process and paradigm, this white paper focuses on the role of IBM's Rational Rapid Developer in facilitating the globalization of Web applications.

Rational Rapid Developer Development Process

The Rational Rapid Developer development process consists of two main steps:

1. *Visually modeling your application using the Rational Rapid Developer Modeling System*
2. *Automatically constructing the code using the Rational Rapid Developer Construction System.*

1. Application Functional Requirements

Functional requirements are the starting point of your application. They are typically captured in the use case models and other document repositories.

2. Import Existing Models and Databases

Although Rational Rapid Developer excels in the development of "greenfield" applications, in which there are no existing assets to incorporate into the new system, the reality of today's enterprise application development is that new applications typically must integrate existing databases, UML models and legacy applications and components of various types.

A frequent requirement in enterprise application development is to harness business functionality that has been modeled in one of the popular UML modeling tools such as IBM Rational Rose. Rational Rapid Developer enables development shops that have adopted UML modeling as a standard to accelerate the implementation phase of the development life cycle by importing UML class diagrams into Rational Rapid Developer and rapidly moving to executable applications. This is accomplished using the open, industry-standard XML Metadata Interchange (XMI) interface, along with a native synchronization feature with Rational Rose and IBM Rational XDE.

Rational Rapid Developer can also save tremendous amounts of time by allowing you to import a database schema from one of the leading relational databases into a class model. This is also known as database reverse engineering. Rational Rapid Developer can also import IMS, VSAM, and CICS files from a legacy environment.

Additionally, any existing Rational Rapid Developer application can be imported into a new application to start the development process. This feature allows you to create your own libraries of "components" that can be reused across the organization. The benefit of these types of components is that they are not tied to any single deployment technology or platform. Instead, they are component models that can be generated at build time into actual code in the desired deployment technology.

3. Rational Rapid Developer Modeling System

Application development begins with functional requirements. The Rational Rapid Developer Modeling System is used to create a detailed model of your application that meets these requirements. Modeling starts at the inception phase of the application when requirements are being gathered. It carries over into the elaboration (design) phase when a high-level model of the application is created. Finally, during the implementation phase, detailed parts of the application are modeled. The modeling involves all aspects of the application: business objects, user interface, messaging and integration. The modeling process does include some hand coding, but this coding is limited to the essential business logic needed to implement business rules and processes, and is typically less than 5% of the total application code.

4. Agile Application Model
The result of the modeling step in Rational Rapid Developer is an agile application model. This model reflects the entire functional needs of your application. At this point, it is a virtual application, about to be generated by the Rational Rapid Developer Construction System.

5. Technical Requirements
It is important to note that the technical requirements of the application are independent of the functional requirements. Technical requirements identify the specific n-tier deployment technologies that are used to deploy your application. Technical requirements drive the design of the deployment model and include the capacity, performance, scalability and availability needs of the application. Technical requirements are specified before you construct the application.

6. Rational Rapid Developer Construction System
The Rational Rapid Developer Construction System constructs and deploys your application model into high-quality, industry-standard n-tier code within minutes. The complexity of n-tier applications is removed from application development and is encapsulated as an engineering discipline in the construction system. Construction is optimized to your specific deployment technologies.

You invoke the Rational Rapid Developer Construction System in all phases of application development, for incremental construction during implementation, and for full construction during system testing and deployment.

7. Deployed Code
The code constructed by Rational Rapid Developer is a rendering of your modeled application, optimized for your selected deployment platform. The code is a constructed artifact of the application but not the actual application. Therefore, modification and maintenance are performed within Rational Rapid Developer and not directly on the constructed code. You can, however, override constructed code components for the various tiers to achieve specific results.

For new versions of your applications, you change your application model and re-construct it to generate new deployment code. After each iteration, the previously deployed code is discarded. As your business needs evolve, you will have a new set of functional requirements to reflect these needs. You use the Rational Rapid Developer Modeling System and modify the application model to meet the new functional requirements. After making all the requisite changes, you re-construct the code, test and debug, and re-deploy the application.

8. Summary

New technologies are rapidly emerging and new versions of existing technologies will continue to be released. To keep pace, businesses need to quickly adapt their existing applications to new technologies and business requirements without losing their investments in current systems. Rational Rapid Developer allows you to design and maintain applications as an application model, rather than at the technology level. Your application model can be used to rapidly redesign, reconstruct and redeploy applications for the latest technologies.

Globalizing a Web Application

Now that you know a little about developing applications with Rational Rapid Developer, let's discuss the globalizing capabilities that have been added to Rational Rapid Developer with the Globalization Model.

There are many issues to contend with when globalizing an existing Web application or when creating a new one. The following sections:

1. *Illustrate the degree to which globalization affects an application.*
2. *Highlight, whenever possible, the role of Rational Rapid Developer in facilitating globalization activities.*

Rational Rapid Developer provides an environment in which you can model and construct the typical set of activities that are associated with globalizing a Web application.

Rational Rapid Developer provides an environment in which you can model and construct the typical set of activities that are associated with globalizing a Web application. You can do this with minimal knowledge of the underlying complexities associated with the implementation platform.

Before Globalizing an Application – Sizing up the Problem

To set the stage for globalization, you must first define the problem. The nature of the problem affects and influences the set of activities and the application design that is required to accomplish your globalization goals. This section discusses general, high-level topics that are just as important as the actual implementation activities.

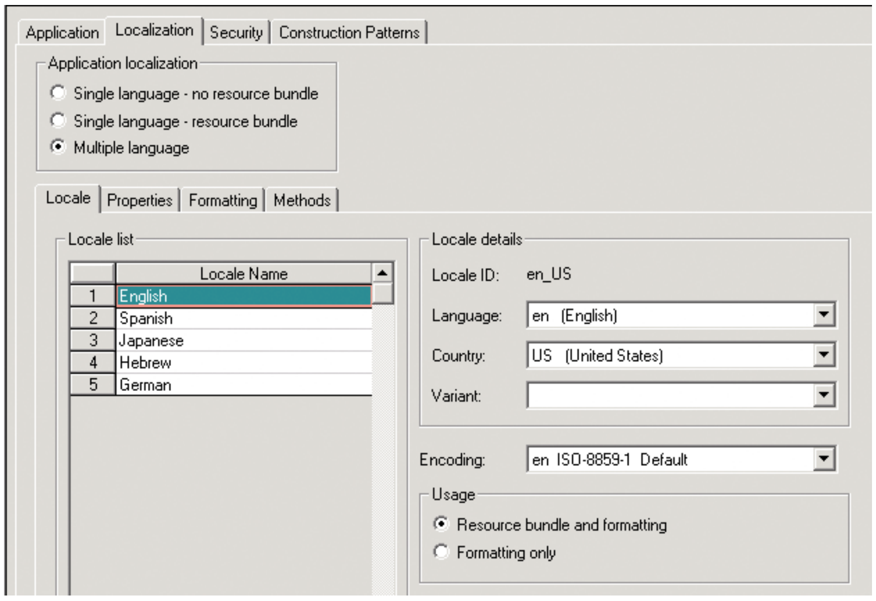
What Languages Will Your Application Need to Support?

The Internet provides worldwide accessibility for your application. You need to consider the languages that must be supported for your application to succeed.

In an enterprise intranet scenario, your company operates in specific countries, so it's easy to determine the languages that your application must support. However, with products such as eBay or Amazon, the reach is more unpredictable and widespread and your application needs to support many languages.

Rational Rapid Developer's Globalization Model provides you with a convenient mechanism to define any number of locale descriptors. Locale descriptors are used to identify cultural characteristics such as language, formatting, sorting, images, and so on. Once defined, they are available throughout your application.

In the following illustration, the Globalization Model shows that the application supports five languages.



Application localization
- ○ Single language - no resource bundle
- ○ Single language - resource bundle
- ● Multiple language

Locale list

	Locale Name
1	English
2	Spanish
3	Japanese
4	Hebrew
5	German

Locale details

Locale ID:	en_US
Language:	en (English)
Country:	US (United States)
Variant:	

Encoding: en ISO-8859-1 Default

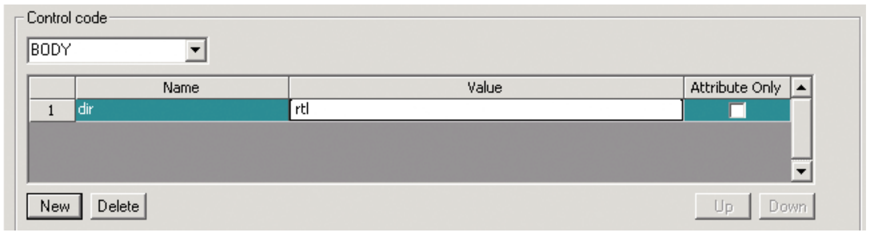
Usage
- ● Resource bundle and formatting
- ○ Formatting only

Figure 1. Globalization Model.

Will You Need to Consider Special Languages?

Some languages pose greater challenges than others. The right-to-left characteristic of Hebrew and Arabic requires special attention to layout and presentation fonts.

In Rational Rapid Developer, the right-to-left attribute can be easily specified in the model without the need to get into the details and complexities of the code. The attribute can be associated with any HTML tag on the page. For example, in the following example, the "dir = rtl" attribute has been applied to the page BODY tag. Rational Rapid Developer will integrate this tag with the HTML produced for the page.



Control code: BODY

	Name	Value	Attribute Only
1	dir	rtl	☐

Figure 2. Setting the Right-to-left Attribute.

Are You Migrating an Existing Application or Are You Creating a New Global-Ready Application?

The complexity of migrating an existing application depends on the degree of thought and effort devoted to globalization when the application was first created. Major effort is required to migrate already existing applications that have not been designed with globalization in mind. From externalizing strings, to providing appropriate calendars, to the creation of formatters and much more, a large amount of code must change.

With Rational Rapid Developer applications, changes are made to the application model and the application is generated, as required. The process of migrating applications in which no thought was given to globalization is much easier in applications developed in Rational Rapid Developer, compared to manually coded applications. Rational Rapid Developer generates Unicode-enabled code, externalizes strings, and generates property files (you won't have to create and manage property files manually). Rational Rapid Developer also generates calendars, formatters, and the requisite code to handle time zones. Virtually any feature that is required for globalization can be modeled in Rational Rapid Developer and the appropriate code generated.

Localization Libraries

Although Java includes built-in internationalization services, your development team will most likely require the use of a library such as IBM's ICU (International Components for Unicode). The library provides a full range of services for supporting internationalization. It provides cross-platform C, C++ and Java APIs. The Java version of ICU builds upon the internationalization features in Sun's Java.

With Rational Rapid Developer, you can leverage powerful libraries such as ICU in custom code that is generated for an application.

Rational Rapid Developer also provides a set of internationalization methods that are at a higher level of abstraction than libraries such as ICU. Using the Rational Rapid Developer internationalization methods reduces complexity while providing platform independence. These methods, and many more, are provided in the Rational Rapid Developer Framework APIs.

What Do You Need to Know About Your Host Development Environment?

The globalization features of the host operating system for your development environment are important to the development of the application. Features such as the ability to set up input locales and the ability to set up browser language preferences will prove to be valuable.

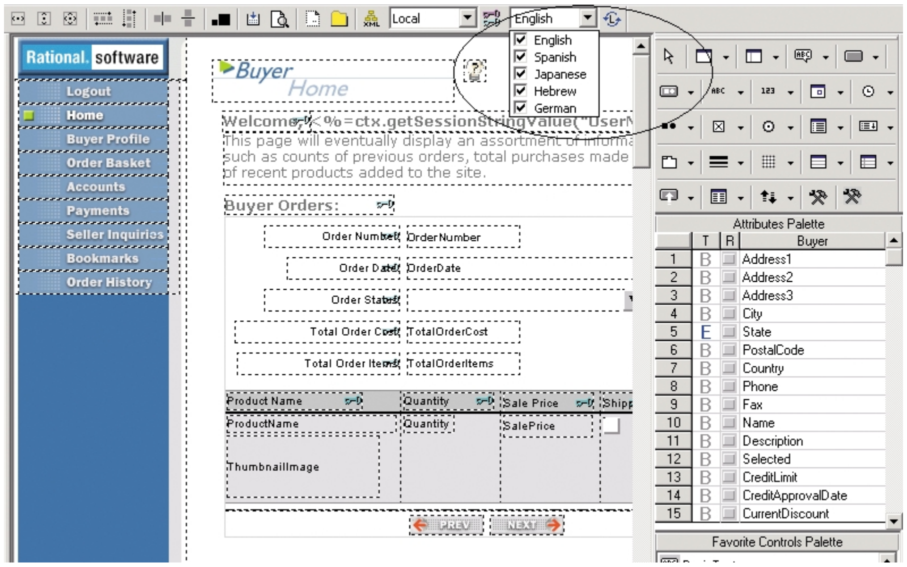
Application Design Strategies

This set of application design strategies will help you plan the implementation of your Web application.

Page Development Strategies

It is often the case that a Web application is created for one culture or locale and then ported to a set of other required cultures. The application is duplicated and then localization changes are made to the copy of the application. While initially appearing easier, consider the problems that are introduced with this methodology. Propagation changes across the set of localized applications will prove to be costly, time consuming and error prone.

Rational Rapid Developer provides an environment that enables you to easily create a single page layout that serves up or produces pages in any language that your application supports. Notice in the following screen capture of the Rational Rapid Developer Page Architect that you can cycle through each language supported by your application. This provides you with a view of the page in each language, at design time. You can easily design the application so that the language in which the page is displayed is selected dynamically at runtime.



Welcome, <%=ctx.getSessionStringValue("User
This page will eventually display an assortment of informa
such as counts of previous orders, total purchases made
of recent products added to the site.

OrderNumber
OrderDate
TotalOrderCost
TotalOrderItems
ProductName
Quantity
SalePrice
ThumbnailImage

	T	R	Buyer
1	B		Address1
2	B		Address2
3	B		Address3
4	B		City
5	E		State
6	B		PostalCode
7	B		Country
8	B		Phone
9	B		Fax
10	B		Name
11	B		Description
12	B		Selected
13	B		CreditLimit
14	B		CreditApprovalDate
15	B		CurrentDiscount

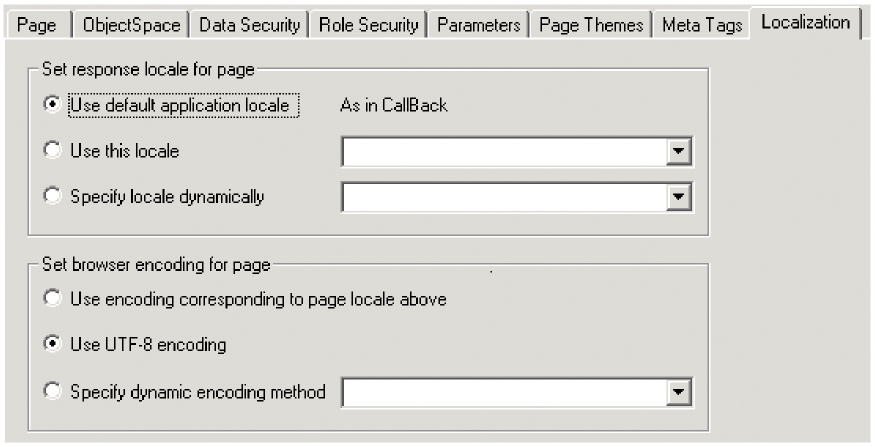
Figure 3. Page Architect, Language Support in Design Time.

This is a unique feature that differentiates Rational Rapid Developer from any other development environment. However, it should be noted that you will not loose the flexibility of creating a page for a specific language, if that is what is required.

Response Locale Strategies

With the ability to define a single, consistent layout for a set of languages, how will you produce the page with the correct language for a given user?

Rational Rapid Developer provides for the ultimate flexibility in defining the response locale for a page. This setting establishes the language for page creation, and thus the language provided to the browser. You can either set the page to a fixed locale or you can define a callback method to set the response locale. While the fixed option provides a static language page, the callback method enables you to define whatever strategy is needed to set up the response locale, and thus the language.



Set response locale for page

(●) Use default application locale — As in CallBack

( ) Use this locale

( ) Specify locale dynamically

Set browser encoding for page

( ) Use encoding corresponding to page locale above

(●) Use UTF-8 encoding

( ) Specify dynamic encoding method

Figure 4. Setting the Response Locale and Encoding.

As an example of one strategy, suppose that your application supports English, Spanish, and Japanese. If the request comes in from the UK, you will respond with an English page. If the request comes from India, you may also want to respond with the English page. You can define a lookup table in the callback method that dynamically determines the response locale.

You could also establish the response locale based on the user preferences that have been stored in a session variable at user login. Yet another strategy could allow the user to change the application language or cultural settings at will.

Browser Encoding Strategies

As demonstrated above, Rational Rapid Developer enables you to establish the language for the page that will be delivered to the browser. This solves only part of the problem. When the page is delivered to the browser, the browser must present the page with the encoding that matches the language delivered. This enables the browser to correctly interpret and present the data. For example, if you try to present Hebrew strings with a Western European encoding, the browser will not be able to interpret the Hebrew strings correctly and the page will not display Hebrew.

As information (e.g., strings, numbers, dates, times) is presented or entered using a browser, the interpretation of the data is based on the encoding that has been set in the browser. You can usually use two types of encodings:

- *Code pages, which handle a limited set of characters (such as Western European)*
- *Unicode, which handles all characters*

Like the response locale, you can design a strategy that sets the correct encoding in the page response. Setting the encoding in the page response in turn sets it in the browser encoding before the page is interpreted and presented.

As shown in the preceding illustration, Rational Rapid Developer provides several ways to set the encoding. The UTF-8 setting represents a Unicode encoding that is a safe way to represent most languages. You can also set the encoding using a callback method (for full flexibility), or by the encoding that has been associated with the page locale.

Database Strategies

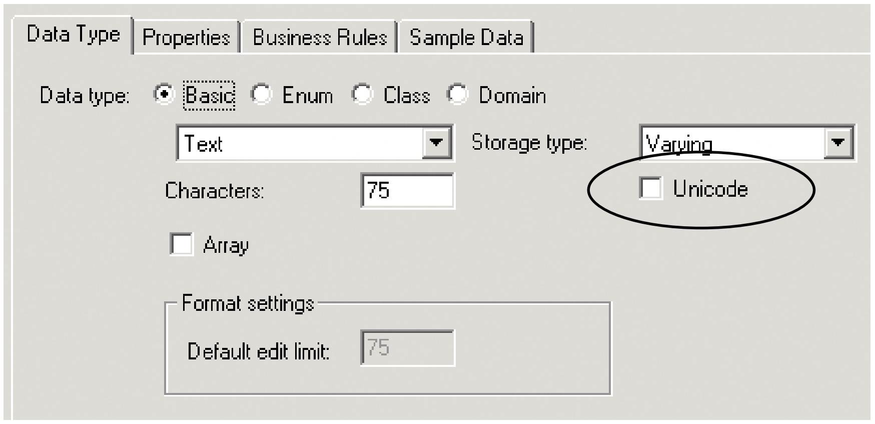
Database design can be grouped into the following categories:

- *Database structure*
- *Data encoding*
- *Data conversion*

Database Structure

Database structure or schema determines how you organize multi-lingual data within an individual database. Will you store data for a specific language in a table devoted to that language, or will you have a column in a table that identifies the language? You may decide to store data from different languages in different databases that mirror each other in structure. The implications of the database structure affect the code that stores, retrieves, and processes data. Maintenance is also affected.

While this is mostly an application database design issue, Rational Rapid Developer provides for properties in the model to specify table attributes that should contain multi-lingual data. By selecting the Unicode checkbox in the text attribute specification, an attribute is created in the database with the Unicode data type specification. For example, the Unicode version of the varchar data type is the nvarchar data type.



Data type: [✓ Basic] Enum Class Domain

Text

Storage type: Varying

Characters: 75

Unicode

Array

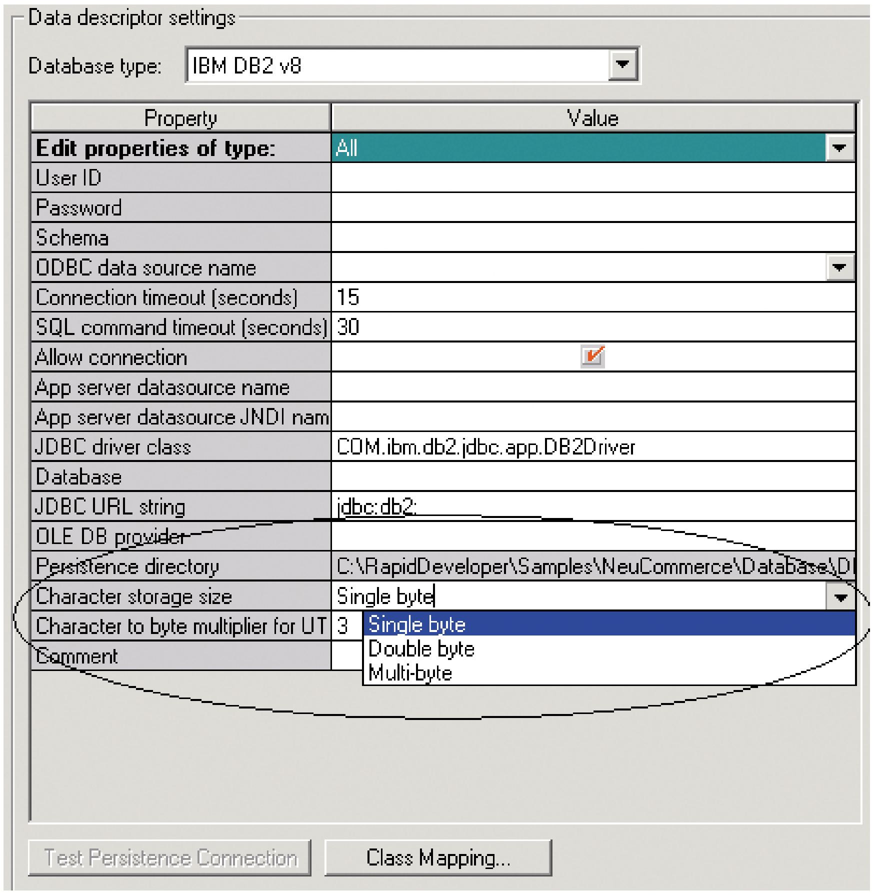
Format settings

Default edit limit: 75

Figure 5. Unicode Data Type Specification.

Data Encoding

Like the encoding that was specified for the browser, the encoding for the database establishes the languages that are correctly stored in the database. Different database vendors handle encoding differently. Rational Rapid Developer understands the encoding capabilities of the supported databases and allows you to make the appropriate encoding selections from the model. The following example of the IBM DB2 database property sheet shows the DB2 properties that are available to define the encoding.



Data descriptor settings

Database type: IBM DB2 v8

Property	Value
Edit properties of type:	All
User ID	
Password	
Schema	
ODBC data source name	
Connection timeout (seconds)	15
SQL command timeout (seconds)	30
Allow connection	✓
App server datasource name	
App server datasource JNDI nam	
JDBC driver class	COM.ibm.db2.jdbc.app.DB2Driver
Database	
JDBC URL string	jdbc:db2:
OLE DB provider	
Persistence directory	C:\RapidDeveloper\Samples\NeuCommerce\Database\DI
Character storage size	Single byte
Character to byte multiplier for UT	3
Comment	

Figure 6. Database Globalization Properties.

Data Conversion

If you are migrating from a single language application to an application that supports multiple languages, you need to consider how to migrate the already existing data without compromising data integrity.

Currency Transaction Strategies

Currency transactions add complexity. The requirements may affect the database structure because of the need to store additional information (e.g., the date and time when a transaction was recorded, the base currency and potentially other attributes of the transaction). Local laws that regulate financial transactions need to be incorporated into any financial processing.

These are mostly application design issues. Rational Rapid Developer facilitates the incorporation of a currency design by allowing you to focus on the application and not the underlying complex technology.

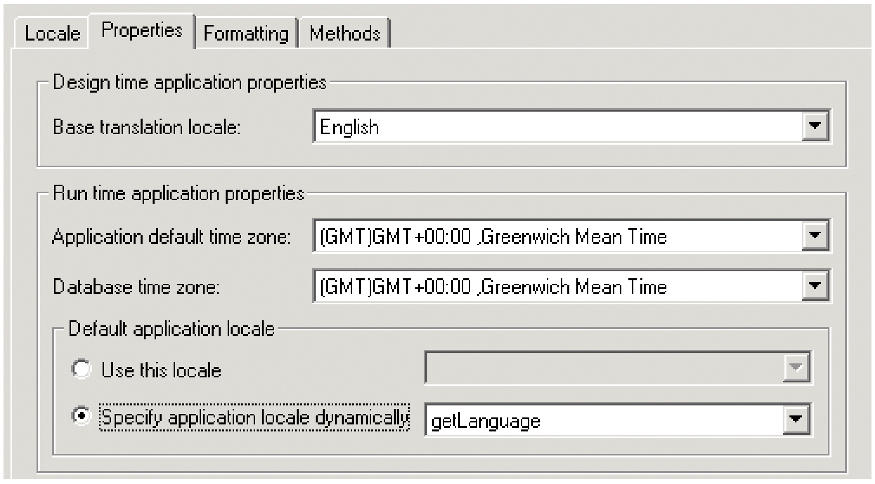
The ability to easily accept Web services into your application also allows you to leverage currency conversion services.

Time Zone Strategies
Presenting and storing date and time data can be difficult in the Web application domain. The time that you display or enter on a Web page depends on the problem that you are trying to solve in the application domain.

What time will you display? Do you display the local time? For example, when you place an order in New York at 9AM Eastern Standard Time, what time will you display for the order date / time on a page displaying that order in California or Hong Kong?

How do you store the data when users in different time zones enter local date / time data? Will you store the local time qualified by some time zone code?

Once you have determined the application design for time zone requirements, you simply provide the Rational Rapid Developer-based application with the time zone for data storage and the default application time zone used to present or enter date / time data. The system does the rest.



Design time application properties

Base translation locale: English

Run time application properties

Application default time zone: (GMT)GMT+00:00 ,Greenwich Mean Time

Database time zone: (GMT)GMT+00:00 ,Greenwich Mean Time

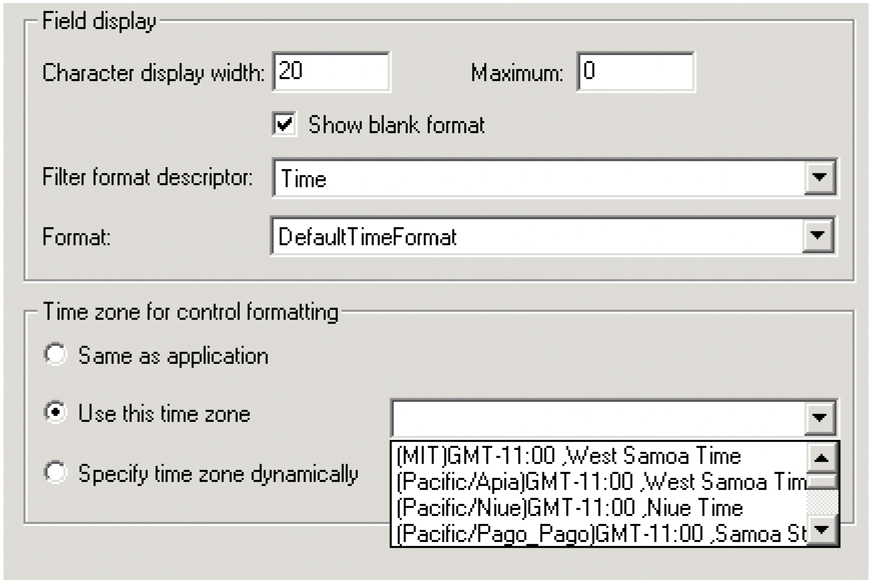
Default application locale

○ Use this locale

● Specify application locale dynamically: getLanguage

Figure 7. Application Time Zone Properties.

You can have a finer granularity of control for time zones by setting a specific time zone for a specific field displayed on a page. This is shown in the following example.



Field display

Character display width: 20 Maximum: 0

[✓] Show blank format

Filter format descriptor: Time

Format: DefaultTimeFormat

Time zone for control formatting

○ Same as application

● Use this time zone

○ Specify time zone dynamically

(MIT)GMT-11:00 ,West Samoa Time
(Pacific/Apia)GMT-11:00 ,West Samoa Tim
(Pacific/Niue)GMT-11:00 ,Niue Time
(Pacific/Pago_Pago)GMT-11:00 ,Samoa St

Figure 8. Time Zone, Field Properties.

Internationalization

Once you have made your strategy decisions, the next step is to complete the implementation. For new applications, this means following a set of rules that dictate how you implement aspects of your application. For existing applications, this involves making modifications to your application that enable you to provide local support, or to localize the application without having to modify code.

In the Rational Rapid Developer paradigm, the part of the definition about not changing code takes on a different meaning since Rational Rapid Developer will generate as much as 95 % of the total application code.

Externalizing Application Strings

When asked what it means to “globalize” an application, most developers will utter the words “externalize your strings” as the first and sometimes only order of business. While there is much more to globalization, externalizing strings is an important part of getting it right.

Of course, if you took the time and effort to externalize your application strings when creating the first version of the application, you would be in a better position to localize. However, it is often the case that globalization is a lower priority than "getting the thing done" and it is overlooked.

For hand-written applications that have not been designed for globalization, this means going through the code and replacing references to strings with functions that fetch the strings from some external string storage mechanism. This can be done manually or through search and replace utilities. This means that you must "touch" all of your code – a process that can introduce errors into your application.

With Rational Rapid Developer, the process of externalizing strings falls into two categories:

1. *Model the strings that need to be translated.*

   Controls such as labels need to present the correct translated string based on the language requested for a page. A collection of translated strings (strings that represent the same meaning in different languages) are assigned a key. The key is then assigned to the control. At runtime, when the page is presented in a given language, the label makes a request for the translated string that matches the requested language.

   You can create keys by first designing a page in a language of choice and specifying the controls that require keys. In the following example, the context menu is on a label and the menu selected creates a key for this control. Notice that other controls that have been associated with keys have a small key indicator on the control.

   Keys with their translations are stored as part of the model within the page definition.

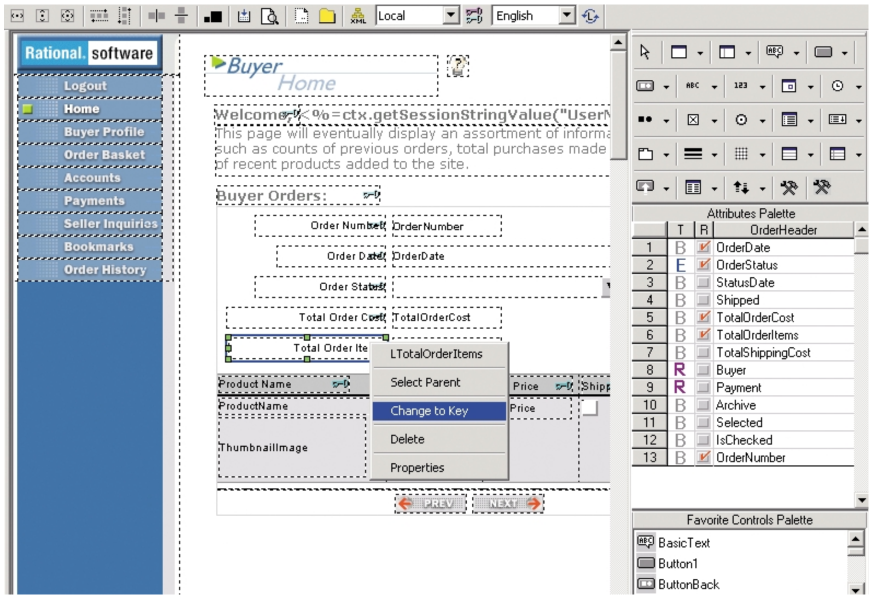
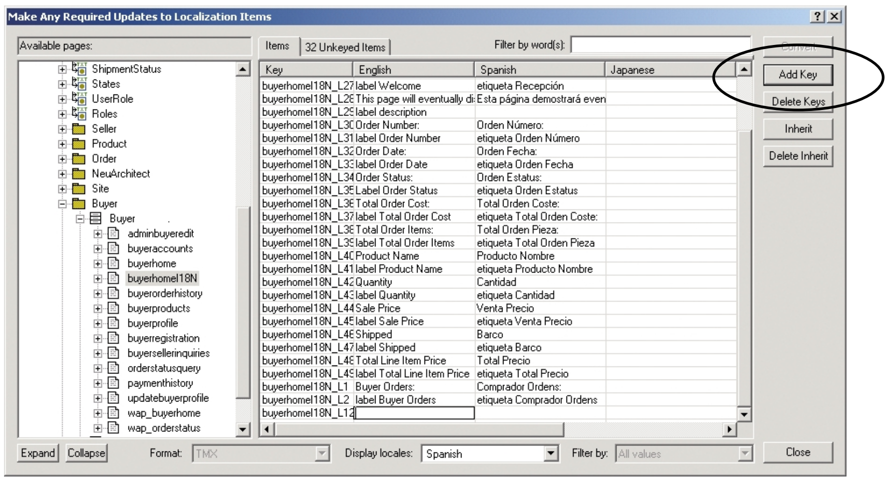


Figure 9. Specifying Individual Text Strings that need Translation.

You can also create keys independently from controls, and then assign the keys to controls.



Key	English	Spanish	Japanese
buyerhomeI18N_L27	label Welcome	etiqueta Recepción	
buyerhomeI18N_L28	This page will eventually di	Esta página demostrará even	
buyerhomeI18N_L29	label description		
buyerhomeI18N_L30	Order Number:	Orden Número:	
buyerhomeI18N_L31	label Order Number	etiqueta Orden Número	
buyerhomeI18N_L32	Order Date:	Orden Fecha:	
buyerhomeI18N_L33	label Order Date	etiqueta Orden Fecha	
buyerhomeI18N_L34	Order Status:	Orden Estatus:	
buyerhomeI18N_L35	Label Order Status	etiqueta Orden Estatus	
buyerhomeI18N_L36	Total Order Cost:	Total Orden Coste:	
buyerhomeI18N_L37	label Total Order Cost	etiqueta Total Orden Coste:	
buyerhomeI18N_L38	Total Order Items:	Total Orden Pieza:	
buyerhomeI18N_L39	label Total Order Items	etiqueta Total Orden Pieza	
buyerhomeI18N_L40	Product Name	Producto Nombre	
buyerhomeI18N_L41	label Product Name	etiqueta Producto Nombre	
buyerhomeI18N_L42	Quantity	Cantidad	
buyerhomeI18N_L43	label Quantity	etiqueta Cantidad	
buyerhomeI18N_L44	Sale Price	Venta Precio	
buyerhomeI18N_L45	label Sale Price	etiqueta Venta Precio	
buyerhomeI18N_L46	Shipped	Barco	
buyerhomeI18N_L47	label Shipped	etiqueta Barco	
buyerhomeI18N_L48	Total Line Item Price	Total Precio	
buyerhomeI18N_L49	label Total Line Item Price	etiqueta Total Precio	
buyerhomeI18N_L1	Buyer Orders:	Comprador Ordens:	
buyerhomeI18N_L2	label Buyer Orders	etiqueta Comprador Ordens	
buyerhomeI18N_L12			

Figure 10. Using the Translation Editor to Create Keyed Strings.

2. *Construct the runtime mechanism that provides access to the externalized strings once they have been translated.*

   Once keys have been created in the model, and you construct the application, Rational Rapid Developer creates resource bundles. The use of resource bundles is a standard mechanism to provide translated strings to an application at runtime. Resource bundles are name/value pairs that define the key and the translated value for a specific language. Resource bundles are created for each language that you have defined in the application Globalization Model.

   The process of externalizing strings involves changes to the model, not to the code. Since the code is generated, the chances for errors are greatly reduced.

Encoding Issues

Setting the encoding involves making decisions about the how data is to be interpreted at various conversion points within the application architecture. The diagram below shows the different elements in an n-tier Web application architecture. The conversion points are:

1. *Between the browser and the JVM (application server) – this is for data and for information passed as query string parameters,*
2. *Between the application server and the database (through JDBC)*
3. *Between the application server and other data sources such as messaging transports.*

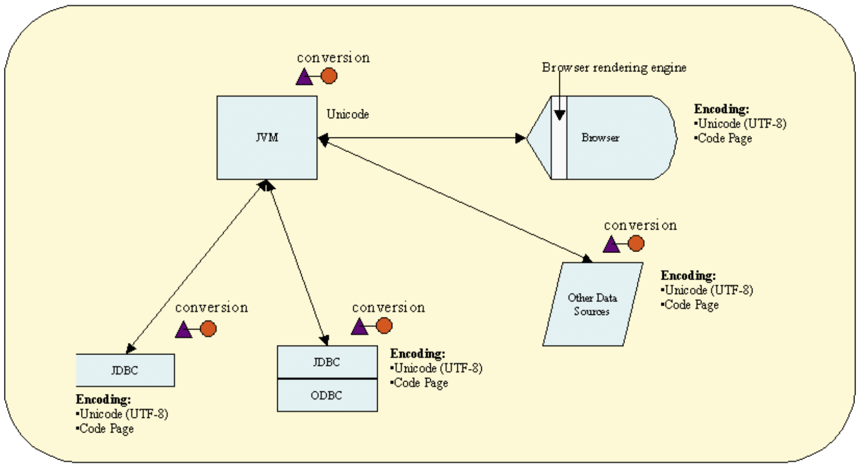
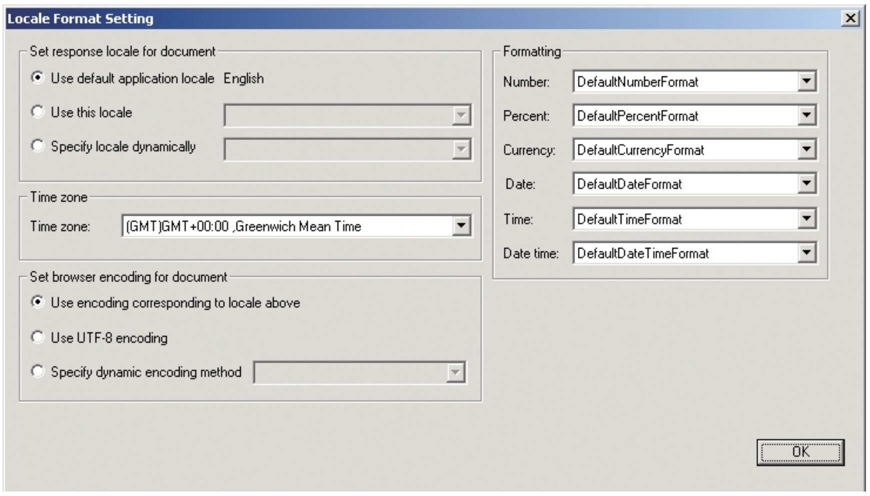


Figure 11. Encoding Conversions within the Application Architecture.

You can see from the diagram that you can use Unicode or code page encoding. Encoding can be set at different points in the application architecture. If you don't take care to match the encodings between conversion points, your data will not be interpreted correctly.

Because there are different technologies involved in Web architectures, setting the encoding may be dependent on other systems. For example, some databases require that you set the encoding when the database is created. XML documents that are sent through message transports require the encoding to be set in the XML header. The following illustration shows the settings that are available for XML documents from the Rational Rapid Developer Message Model.


Locale Format Setting

Set response locale for document
- [✓ Use default application locale] English
- Use this locale
- Specify locale dynamically

Time zone
Time zone: (GMT)GMT+00:00 ,Greenwich Mean Time

Set browser encoding for document
- [✓ Use encoding corresponding to locale above]
- Use UTF-8 encoding
- Specify dynamic encoding method

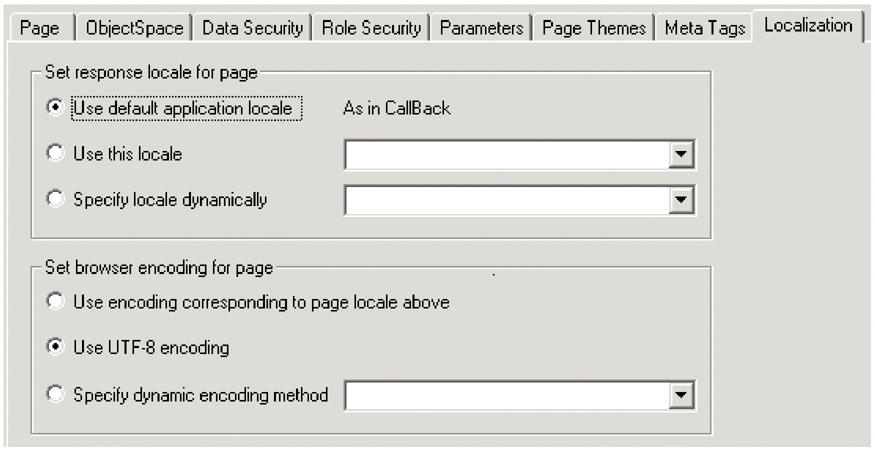
Formatting

Number:	DefaultNumberFormat
Percent:	DefaultPercentFormat
Currency:	DefaultCurrencyFormat
Date:	DefaultDateFormat
Time:	DefaultTimeFormat
Date time:	DefaultDateTimeFormat

Figure 12. Settings for Message Based XML Documents.

The data that is passed between the browser and the application server is the data that makes up the page, and optionally parameters that are passed on the URL. Since this data can be in multiple locales, you need to be able to set an encoding property for the data and for the parameters on the URL.

To set the data encoding, you need to set it on the browser. For example, you can set the encoding for Microsoft Internet Explorer using an Encoding submenu on the View menu. The trick, however, is that you want the page being delivered to the browser to set the encoding for the browser, not the end user. Rational Rapid Developer enables you to control this setting with the encoding property on the page property tab, as shown in the following illustration.



Set response locale for page

[✓ Use default application locale] As in CallBack

Use this locale

Specify locale dynamically

Set browser encoding for page

Use encoding corresponding to page locale above

[✓ Use UTF-8 encoding]

Specify dynamic encoding method

Figure 13. Setting the Page Encoding Property.

Rational Rapid Developer sets the encoding on the URL automatically.

User Interface Issues

A common approach to localization of Web pages is to create a new page for each language. Unfortunately, this creates a large number of pages that need to be maintained. When a modification is made to one page, it will most likely need to be made to the corresponding pages in different languages. This is a maintenance nightmare and an error-prone methodology.

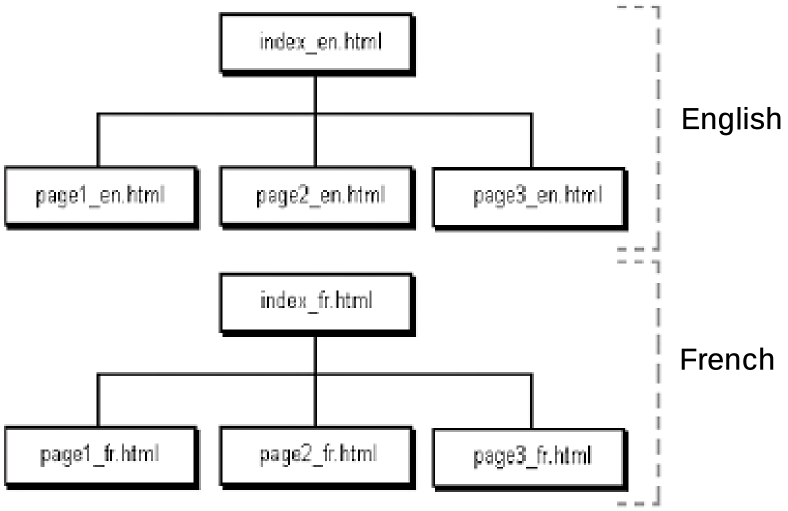


Figure 14. Typical Approach: One Page Per Language.

Your internationalization goals should be to be able deliver multiple languages from a single page. This greatly reduces the number of pages and the amount of maintenance required when modifying the application. Rather than making similar modifications for each page that represents a given language, you make a set of changes and the application delivers the page in the requested language.

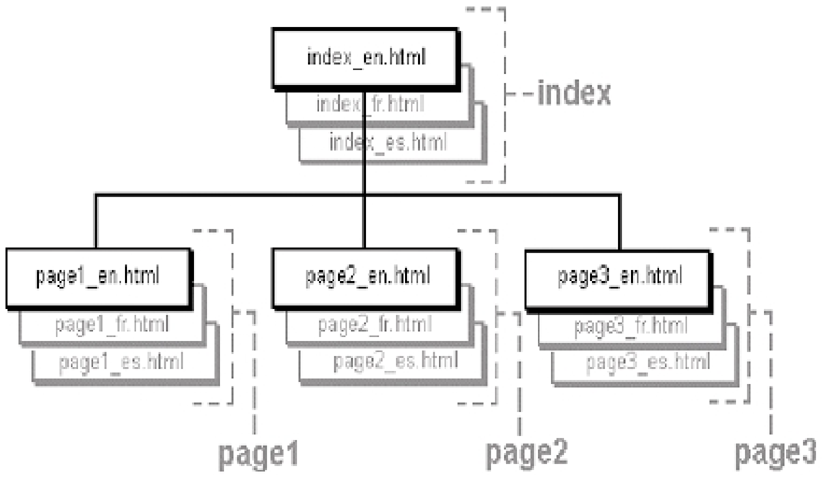


Figure 15. Optimal Approach: Serve Up Multiple Languages from a Single Page.

Rational Rapid Developer facilitates this more efficient methodology by allowing you to view the same page in different languages. During runtime, the page appropriate for the request is delivered to the browser.

Formatting Issues

Formatting in a globalized application can be very complex. The goal is to provide one set of code that reflects whatever locale or cultural settings required for an instance of a page that is being delivered to a browser.

Java has a robust API to handle this situation; however, navigating through all of the classes and learning how to use the API can be tedious. Rational Rapid Developer provides you with a Formatting Model that allows you to create format descriptors for each locale and each formatting type. Formatters are available for numbers, currency, percent, date, time, and date-time.

The Rational Rapid Developer Formatting Model is shown in the following illustration. You can see that every locale has a default format for each formatting type. You can also create your own format descriptor for any format type.

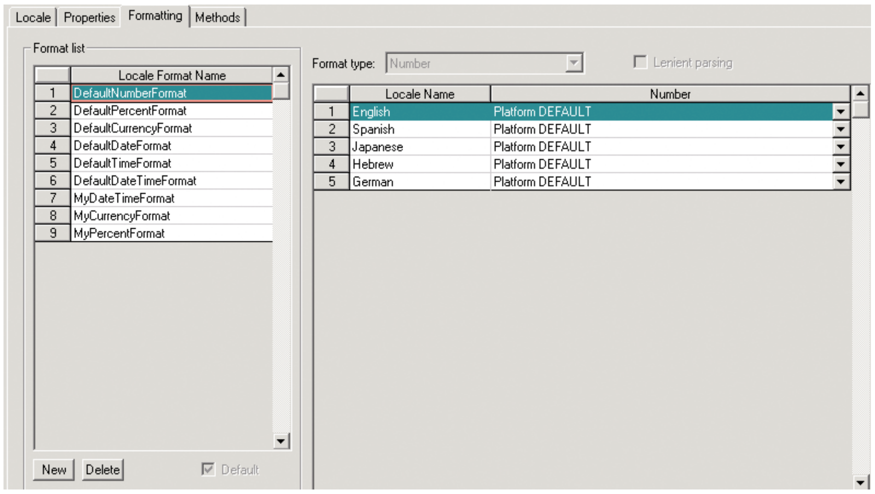


Figure 16. Formatting Model.

Locale | Properties | Formatting | Methods

Format list

	Locale Format Name
1	DefaultNumberFormat
2	DefaultPercentFormat
3	DefaultCurrencyFormat
4	DefaultDateFormat
5	DefaultTimeFormat
6	DefaultDateTimeFormat
7	MyDateTimeFormat
8	MyCurrencyFormat
9	MyPercentFormat

New | Delete | Default

Format type: Number — Lenient parsing

	Locale Name	Number
1	English	Platform DEFAULT
2	Spanish	Platform DEFAULT
3	Japanese	Platform DEFAULT
4	Hebrew	Platform DEFAULT
5	German	Platform DEFAULT

The Formatting Model enables you to establish a set of format descriptors. You need to use these descriptors to control the presentation or the data entry format for specific data items on a page.

The following illustration shows how you can do this within the property sheet for a control. By simply assigning a format descriptor to a control, Rational Rapid Developer will know the formatters, calendars, and other items to create when constructing the underlying code.

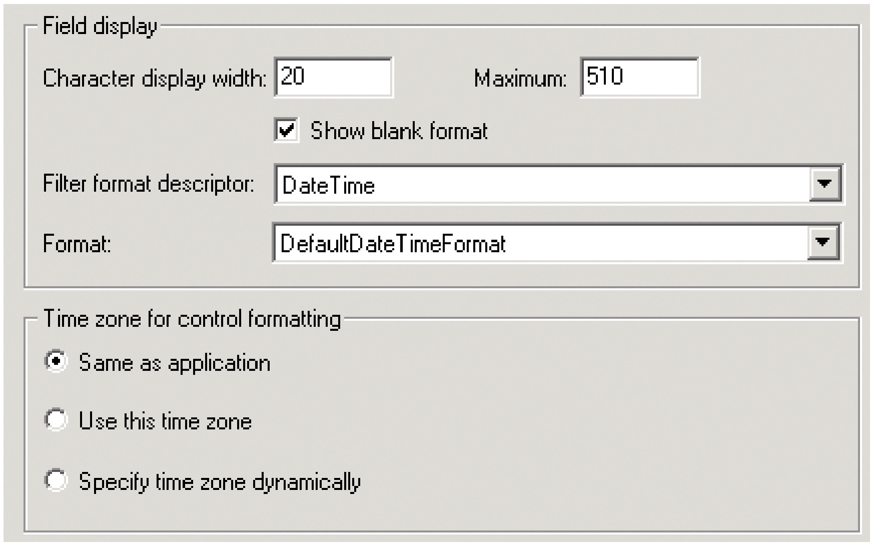


Figure 17. Setting the Format for a Control.

Field display

Character display width: 20 Maximum: 510

[✓] Show blank format

Filter format descriptor: DateTime

Format: DefaultDateTimeFormat

Time zone for control formatting

[●] Same as application

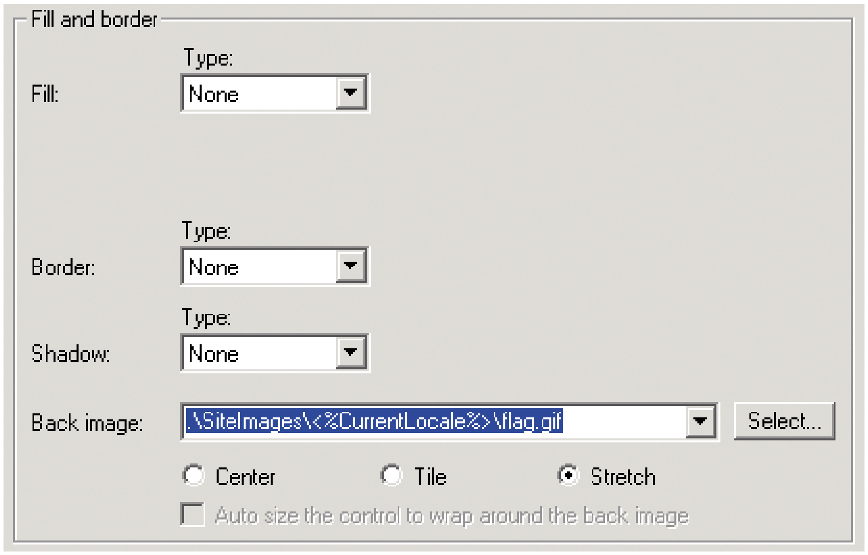
○ Use this time zone

○ Specify time zone dynamically

Besides presentation, you should also be aware that formatting descriptors are used to set the format for data that is being entered into the page and its subsequent validation. Validation can be in the form of client validation through Client Side JavaScript and Server side validation.

Managing Images

Images must also be handled correctly in a multi-lingual environment. Rational Rapid Developer provides a flexible yet easy way to prepare for the localization of images. You can specify the relative location of a set of images that need to be presented on a page, based on the response locale for the page. Notice in the following illustration that the relative path of the image is specified with a macro - <%CurrentLocale%>. This macro is evaluated at design time or at runtime. This gives you the ability to create locale-based folders that store sets of localized images.



Fill: None
Border: None
Shadow: None
Back image: .\SiteImages\<%CurrentLocale%>\flag.gif
( ) Center ( ) Tile (●) Stretch
☐ Auto size the control to wrap around the back image

Figure 18. Specifying Images Dynamically Based on Language.

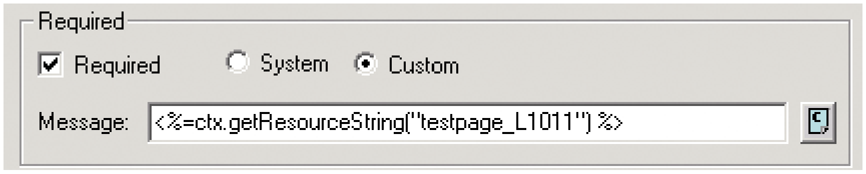
Managing Error Messages

You can't complete the globalization of your application without preparing error messages that can be presented in multiple languages.

Rational Rapid Developer facilitates the globalization of error messages in several ways:

- *By providing a model-based mechanism for internationalizing client-side JavaScript messages for data entry forms.*

The following illustration shows the properties that can be set for a required data entry field. The string associated with the key "testpage_L1011" will be delivered as a client side dialog box when you attempt to submit a form without entering something for the required field. Rational Rapid Developer creates and delivers the dialog box localized for the response locale for the page at the time the error occurs.


Required

[✓] Required ○ System ● Custom

Message: <%=ctx.getResourceString("testpage_L1011") %>

Figure 19. Specifying Error Messages Dynamically Based on Language.

- *By providing a general interface that can be used in any place in your custom methods.*

```
ctx.getResourceString("G12")
```

The method returns a localized string based on the page response locale.

Other Internationalization Features

Rational Rapid Developer offers a robust set of features to deal with the challenges presented by globalization. Other available features include:

Sorting
Selection
Section 508 Help
Help
Multiple Languages on a single page
Internationalized enumerations
Localized auto-constructed images.

Localization

Once you have prepared the application for different languages, you will need to localize the application.

The process of localizing an application can be managed through Globalization Management Systems (GMS). These systems manage the process of localization through a pre-defined workflow. The translation is one step in the workflow.

Rational Rapid Developer provides several features that greatly facilitate the translation process. They are:

- *The ability to view translation and localization impact at design time*
- *The management of translation strings into your application*
- *The granularity offered for the integration of translations*

Design Time vs. Runtime Testing

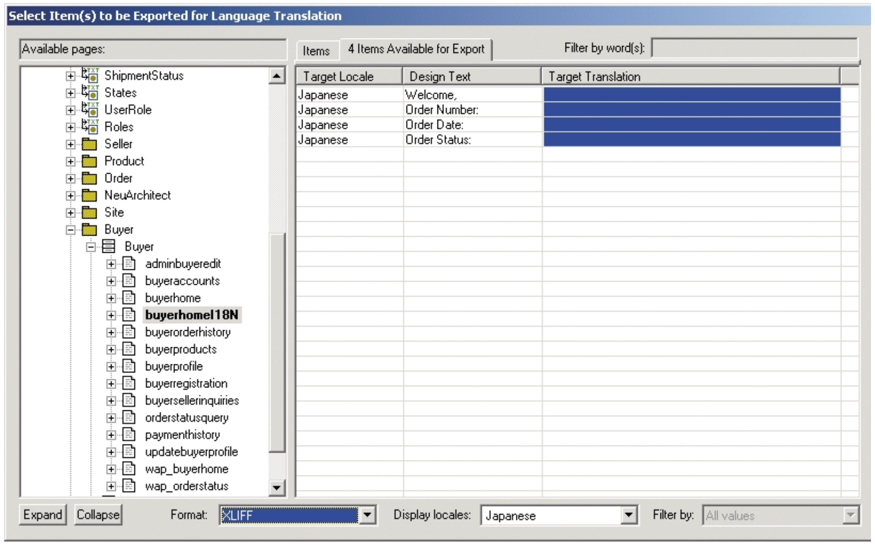
When developing a localized application, the designer / developer / tester / translator (members of the development team) must be able to test the pages in multiple languages. Not only are they testing the pages for proper translation, but they are also focusing on layout consistency issues. Often, the developer must wait until the page can be viewed at runtime to catch any layout conflicts.

With Rational Rapid Developer, you can detect layout conflicts at design time by toggling between different languages in the Page Architect page-authoring module. Design-time review saves time by catching layout defects earlier in the process.

Translation

To translate your application strings to multiple languages, Rational Rapid Developer offers a convenient method of exporting keyed strings from your application to a variety of formats. The resulting file is sent to translators.

In the following illustration, four strings that need to be translated into Japanese are selected in a specific page. The strings will be sent to the translators in XLIFF format. You also can select TMX, Excel, or CSV export formats.



Select Item(s) to be Exported for Language Translation

Available pages:

- ShipmentStatus
- States
- UserRole
- Roles
- Seller
- Product
- Order
- NeuArchitect
- Site
- Buyer
  - Buyer
    - adminbuyeredit
    - buyeraccounts
    - buyerhome
    - **buyerhomeI18N**
    - buyerorderhistory
    - buyerproducts
    - buyerprofile
    - buyerregistration
    - buyersellerinquiries
    - orderstatusquery
    - paymenthistory
    - updatebuyerprofile
    - wap_buyerhome
    - wap_orderstatus

Items | 4 Items Available for Export | Filter by word(s):

Target Locale	Design Text	Target Translation
Japanese	Welcome,	
Japanese	Order Number:	
Japanese	Order Date:	
Japanese	Order Status:	

Expand | Collapse | Format: XLIFF | Display locales: Japanese | Filter by: All values

Figure 20. Exporting Strings to be Translated.

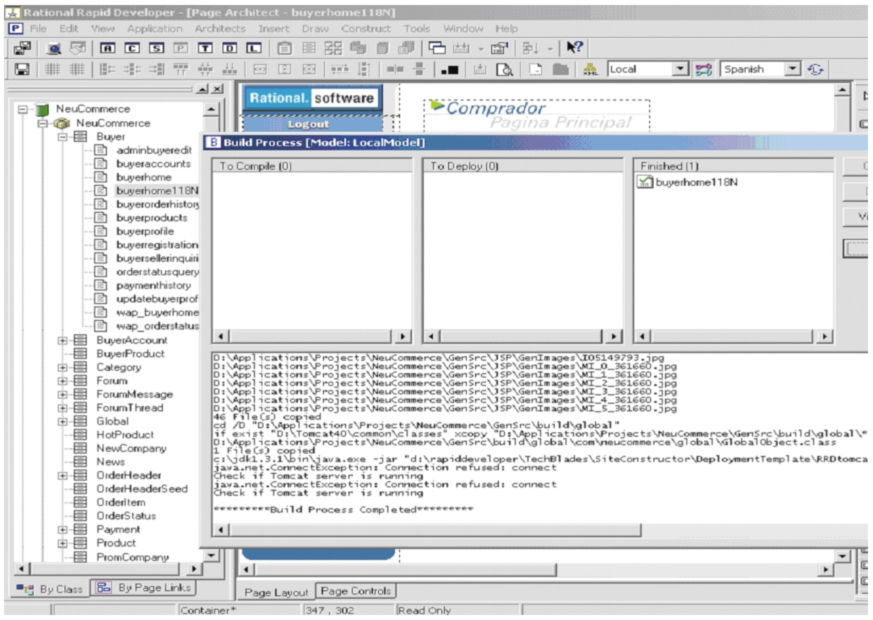
Once the translated files are returned from the translators, you simply place it in a “Translated” directory in the application directory. Rational Rapid Developer automatically imports the strings and integrates them back into the model. You can inspect the translation and optionally select the translated strings that you want to incorporate back into the model.

Translation Granularity

In most translation scenarios, you create an application in one language and then send out all the application strings to several translators. Since it is so easy to prepare strings to be translated and to incorporate the translations back into the model, Rational Rapid Developer gives you the choice to translate on a page-by-page basis at any time during your development process. Your development process can include a check to see if a page is ready for translation. As long as the translators are available, they can provide the translations as different parts of the application are ready for translation. There is no need to wait until the entire application is completed. There may be reasons to wait for the complete application to be finished but at least with Rational Rapid Developer, you are not trapped into this workflow due to the complexity of localizing hand-coded applications. You now have a choice.

Construction

At any point in the process you can construct your entire application or individual pages, messages, components, or Web services. Rational Rapid Developer takes the model specifications and constructs optimized code for the target platform.



```
D:\Applications\Projects\NeuCommerce\GenSrc\JSP\GenImages\I05149793.jpg
D:\Applications\Projects\NeuCommerce\GenSrc\JSP\GenImages\MI_0_361660.jpg
D:\Applications\Projects\NeuCommerce\GenSrc\JSP\GenImages\MI_1_361660.jpg
D:\Applications\Projects\NeuCommerce\GenSrc\JSP\GenImages\MI_2_361660.jpg
D:\Applications\Projects\NeuCommerce\GenSrc\JSP\GenImages\MI_3_361660.jpg
D:\Applications\Projects\NeuCommerce\GenSrc\JSP\GenImages\MI_4_361660.jpg
D:\Applications\Projects\NeuCommerce\GenSrc\JSP\GenImages\MI_5_361660.jpg
46 File(s) copied
cd /D "D:\Applications\Projects\NeuCommerce\GenSrc\build\global"
if exist "D:\Tomcat40\common\classes" xcopy "D:\Applications\Projects\NeuCommerce\GenSrc\build\global\*
D:\Applications\Projects\NeuCommerce\GenSrc\build\global\com\neucommerce\global\GlobalObject.class
1 File(s) copied
c:\jdk1.3.1\bin\java.exe -jar "d:\rapiddeveloper\TechBlades\SiteConstructor\DeploymentTemplate\RRDtomca
java.net.ConnectException: Connection refused: connect
Check if Tomcat server is running
java.net.ConnectException: Connection refused: connect
Check if Tomcat server is running

*********Build Process Completed*********
```

Figure 21. Constructing the Translated Page.

Conclusions

There are many issues involved in application globalization. Rational Rapid Developer addresses these issues and makes it far easier to create and maintain applications that can be delivered in any number of languages. The Rational Rapid Developer model-driven, architected RAD development environment ensures productivity in design and implementation. It provides an abstraction from the complexity involved not only in building an n-tier Web application, but also in building an application that supports multiple languages.