Working in Multiple Locales
using Actuate Basic Technology
Contents

About Working in Multiple Locales using Actuate Basic Technology. . . . v

Chapter 1
Introduction to locales ................................................................. 1
About locales .............................................................................. 2
About character sets ................................................................... 2
  About Unicode ........................................................................ 2
  About code pages ................................................................... 3
  About the Hong Kong Supplementary Character Set (HKSCS) ........... 3
About character entry and display ................................................. 3
About fonts ................................................................................ 3
About date, time, currency, and number formats .............................. 4
  Understanding date and time formats ......................................... 4
  Understanding currency and number formats .............................. 4
About calendars .......................................................................... 5
About collation sequences ............................................................ 5
About case mapping ................................................................... 6

Chapter 2
Formatting report data for multiple locales ................................. 7
About locale precedence ............................................................... 8
Running reports with Actuate e.Report Designer Professional .......... 8
About supported locales ............................................................... 9
Using the locale map ................................................................... 11
  Understanding localemap.xml .................................................. 11
  About the AB tags ................................................................ 20
Using localemap.xml ................................................................... 21
  Using localemap.xml with Actuate iServer ................................. 21
  Using localemap.xml with Actuate web-based products ............... 21
  Using localemap.xml with e.Report Designer Professional .......... 22
Creating a custom locale ............................................................. 22
Specifying a locale with Actuate Information Console ................... 22
Setting a report’s locale ............................................................... 23
Setting the Locale property .......................................................... 23
Setting different locales for generating, viewing, and printing a report 24
  Using GetFactoryLocale( ) ..................................................... 24
  Using GetViewLocale( ) ......................................................... 25
  Using GetPrintLocale( ) ......................................................... 25
Setting different locales for generating and viewing a report .......... 25
Parsing strings with Actuate Basic functions ................................. 26
Using locale-independent parsing ............................................. 26
  Using the ParseNumeric function ....................................... 26
  Using the ParseDate function .......................................... 27
Understanding locale-dependent parsing ..................................... 27
  Understanding functions that operate on numeric expressions ..... 28
  Understanding functions that operate on date expressions ...... 29
Formatting dates, times, currency, and numbers ................................. 30
  Formatting dates and times .............................................. 30
  Formatting currency and numbers ..................................... 31
  Using a pre-Euro currency symbol .................................... 32
Getting the locale name and locale attributes ................................. 32
  Understanding GetLocaleName( ) ....................................... 32
  Understanding GetLocaleAttribute( ) .................................. 32
  Using GetLocaleName( ) and GetLocaleAttribute( ) ............. 33
Understanding parameter handling ........................................... 34
Designing Japanese reports .................................................... 34
Using a localized sfdata database and externalized strings ......................... 35
  About the localized sfdata database .................................. 35
  About the externalized strings ....................................... 36
  About the hash array library .......................................... 36
  Processing the text file ............................................... 37
  Retrieving the correct French string from the hash array ........ 37

Chapter 3
Understanding report encoding .................................................. 39
About report encoding .......................................................... 40
  Setting the ReportEncoding property .................................. 43
  Understanding the Language variable .................................. 43
Running reports with e.Report Designer Professional ............................. 44
Working with encoding and Actuate Basic functions ......................... 44
  Working with functions that operate on UCS-2 character codes .... 45
    Using the AscW function ............................................. 45
    Using ChrW and ChrW$ functions ................................... 45
    Using StringW and StringW$ functions ............................. 45
  Working with functions that operate on code page character codes .... 45
    Using the Asc function ............................................. 45
    Using Chr and Chr$ functions ...................................... 46
    Using the String$ function ......................................... 46
  Working with functions that operate on byte length ..................... 46
Using the Actuate Basic Open statement ..................................... 47
About Actuate Basic functions that require conversion to code page ........... 48
  About Environ and Environ$ functions ................................ 48
  About the Shell function .............................................. 48
Working in Multiple Locales using Actuate Basic Technology provides information for report developers and system administrators who design and deploy Actuate Basic report applications in locales other than U.S. English.

e.Report Designer Professional is supported on Windows 7, Windows Vista, and Windows XP platforms. The default installation location on Windows 7 and Windows Vista platforms is \Program Files (x86)\Actuate11\eRDPro. The default installation location on Windows XP platforms is \Program Files\Actuate11\eRDPro. In paths throughout this document, the default installation location for e.Report Designer Professional is represented by <eRDPro_HOME>. Illustrations show the default installation path on Windows XP platforms.

Likewise, Actuate iServer is also supported on Windows 7, Windows Vista, and Windows XP platforms. The default installation location on Windows 7 and Windows Vista platforms is \Program Files (x86)\Actuate11\iServer. The default installation location on Windows XP platforms is \Program Files\Actuate11 \iServer. Throughout this document, paths are given for Windows XP and UNIX platforms only.

Working in Multiple Locales using Actuate Basic Technology includes the following chapters:

■ About Working in Multiple Locales using Actuate Basic Technology. This chapter provides an overview of this guide.

■ Chapter 1. Introduction to locales. This chapter explains what a locale is and describes the various elements that make up a locale.
Chapter 2. Formatting report data for multiple locales. This chapter describes the locale map and explains how the formats for Actuate Basic report data are determined.

Chapter 3. Understanding report encoding. This chapter explains how Actuate determines the encoding to use when communicating with other programs.

Chapter 4. Using fonts in reports with multiple locales. This chapter explains how Actuate obtains the font metrics for the fonts used in an Actuate Basic report.

Chapter 5. Designing reports with right-to-left orientation. This chapter describes how to design Actuate Basic reports with right-to-left orientation.

Appendix A. Locale codes. This appendix lists the locale codes used in Actuate Basic code.
Chapter 1, Introduction to locales

Introduction to locales

This chapter contains the following topics:

■ About locales
■ About character sets
■ About character entry and display
■ About fonts
■ About date, time, currency, and number formats
■ About calendars
■ About collation sequences
■ About case mapping
About locales

Multinational corporations throughout the world deploy Actuate applications across cultural boundaries. Each culture has a set of conventions for entering, displaying, and sorting data. This set of conventions is called a locale.

A locale specifies the following:
- Code page
- Character entry and display
- Fonts
- Format for dates, times, currency, and numbers
- Calendar
- Collation sequence
- Case mapping

Internationalization is the process of making an application work correctly in multiple locales.

About character sets

A character set is a mapping of specific characters to code points. For example, in most character sets the letter A is mapped to the hexadecimal value 0x21. Most languages use single-byte character sets. Chinese, Japanese, and Korean use multibyte character sets.

About Unicode

Unicode is a character set that contains nearly every character from every modern language and several ancient languages. If a file is encoded using Unicode, the file can include any combination of languages. The most commonly used Unicode encoding schemes are UCS-2 and UTF-8. UTF-8 contains more characters than UCS-2.

With UCS-2 encoding, every character is two bytes in length, including ASCII characters. For example, the letter A is two bytes. Its hexadecimal value is 0x0021. Actuate uses UCS-2 encoding for proprietary file types such as ROD, ROX, and ROI.

With UTF-8 encoding, characters vary in length from one to six bytes. ASCII characters are one byte, just as they are in other character sets. For example, the letter A is one byte. Its hexadecimal value is 0x21.
About code pages

Different platforms support different code pages. Every code page contains the ASCII characters in the first 128 positions. Each code page also contains characters from one or more other languages. For example, Microsoft Windows code page 1252 contains the ASCII characters as well as characters from many Western European languages.

A file is ordinarily encoded using a single code page that contains characters from a specific set of languages. For this reason, certain combinations of languages cannot be included in a single file unless the file uses Unicode encoding. For example, French and Japanese cannot be included in a single file unless the file uses Unicode encoding. English, however, can be included in a file in combination with any other language.

About the Hong Kong Supplementary Character Set (HKSCS)

Traditional Chinese character sets such as Big5 (Windows code page 950) do not contain many characters commonly used in Hong Kong. The Hong Kong Supplementary Character Set (HKSCS) contains approximately 5000 characters that are used in Hong Kong but not included in Big5. Many of these characters are included in Unicode. For more information about the Hong Kong Supplementary Character Set, navigate to one of the following URLs:

http://www.microsoft.com/hk/hkscs/

About character entry and display

Character entry and display depend on the language. Many languages, including English, read from left to right. Middle Eastern and North African languages read from right to left. Traditionally, Chinese, Japanese, and Korean read from top to bottom and right to left. Computers, however, read Chinese, Japanese, and Korean from left to right.

About fonts

A font specifies how characters are displayed and printed. Most fonts work with only one code page. On Microsoft Windows, some fonts, called Big Fonts, work with more than one code page. Arial, Courier New, Lucida Console, Lucida Sans Unicode, and Times New Roman are all Big Fonts. Arial, for example, includes Arial (Western), Arial (Cyrillic), and Arial (Greek). Big Fonts can support many more languages than conventional fonts.
About date, time, currency, and number formats

The format used for dates, times, currency, and numbers depends on the locale.

Understanding date and time formats

Date and time data can include one or more of the following elements:

- The name of the month
- The name of the day of the week
- BC, AD, BCE, B.C., A.D., or B.C.E.
- AM, PM, A.M., or P.M.

The language and sequence of these elements depend on the locale.

Example

In the U.S. English locale, a date can be represented as follows:

Monday, May 17, 1999

In the standard French locale, the same date can be represented as follows:

lundi 17 mai 1999

Understanding currency and number formats

Currency and number data can include one or more of the following elements:

- The ISO currency symbol
- The local currency symbol
- The decimal separator
- The group (thousands) separator

The characters used for these elements depend on the locale.

Examples

In the U.S. English locale:

- The local currency symbol is $.
- The group (thousands) separator is the comma.
- The decimal separator is the period.

Currency data formatted with the local currency symbol appears as follows:

$12,345.67

In the standard French locale:

- The local currency symbol is €.
The group (thousands) separator is the space.
The decimal separator is the comma.
Currency data formatted with the local currency symbol appears as follows:

12 345,67 €

About calendars

The calendar used in many locales is the Gregorian calendar. Other calendars in use throughout the world include:

- Arabic Hijrah
- English Hijrah
- Japanese Imperial
- Persian
- ROC Official (Republic of China)
- Thai Buddha

About collation sequences

Very often, computers sort characters based on the characters’ binary values in the character set. This sort order is called a binary sort. A binary sort does not always yield a result that is consistent with the locale’s language. A sort order that is consistent with the locale’s language is called a linguistic sort or collation sequence.

Example

Consider the three characters Ä, B, and Z. A binary sort using the ISO 8859-1 character set yields the following result:

B
Z
Ä

This sort order is correct for Swedish but not for German. For German, the correct sort order is as follows:

Ä
B
Z

Actuate does not support linguistic sorting.
About case mapping

The uppercase or lowercase forms of letters can depend on the locale.

**Example**

In the standard German locale, the uppercase form of the German sharp s

ß

is

SS

In the German (Austria) locale, the uppercase form of the German sharp s

ß

is

SZ
Chapter 2

Formatting report data for multiple locales

This chapter contains the following topics:

- About locale precedence
- Running reports with Actuate e.Report Designer Professional
- About supported locales
- Using the locale map
- Specifying a locale with Actuate Information Console
- Setting a report’s locale
- Parsing strings with Actuate Basic functions
- Formatting dates, times, currency, and numbers
- Getting the locale name and locale attributes
- Understanding parameter handling
- Designing Japanese reports
- Using a localized sfdata database and externalized strings
About locale precedence

The format for Actuate Basic report data is determined by, from highest precedence to lowest precedence:

- The locale used by the Actuate Basic Format and Format$ functions.
- The report’s locale.
- The view request locale.
- The iServer System’s default locale. For more information about setting the default iServer System locale parameter, see Configuring BIRT iServer.
- The default locale defined in localemap.xml.
- The US English locale.

Running reports with Actuate e.Report Designer Professional

When you run a report using Actuate e.Report Designer Professional, the locale precedence is different from the locale precedence on the iServer System.

On the desktop, the format for report data is determined by, from highest precedence to lowest precedence:

- The locale used by the Actuate Basic Format and Format$ functions.
- The report’s locale.
- The default locale specified on the General page of the Options dialog.
- The default locale specified when the product is installed.
- The default locale defined in localemap.xml.

How to specify the default locale for e.Report Designer Professional

1 Choose Tools ➔ Options.
2 In Options, choose General.
3 In Default locale, choose a locale from the drop-down list, as shown in Figure 2-1. Choose OK.

Your display is not refreshed until you take further action.

If you want the settings for e.Report Designer Professional’s default locale to match the Windows Regional Settings, you must modify localemap.xml. For
example, if you want the Long date format for the French (France) locale to match the Long date format for the Windows French (Standard) locale, you must modify the Long date format for the French (France) locale in localemap.xml.

Figure 2-1  Selecting a default locale

About supported locales

Actuate Basic reports support the following locales:

- Albanian
- Arabic (Algeria)
- Arabic (Bahrain)
- Arabic (Egypt)
- Arabic (Iraq)
- Arabic (Jordan)
- Arabic (Kuwait)
- Arabic (Lebanon)
- Arabic (Libya)
- Arabic (Morocco)
- Arabic (Oman)
- Arabic (Qatar)
- Arabic (Saudi Arabia)
- Arabic (Syria)

(continues)
<table>
<thead>
<tr>
<th>Arabic (Tunisia)</th>
<th>German (Liechtenstein)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic (U.A.E.)</td>
<td>German (Switzerland)</td>
</tr>
<tr>
<td>Arabic (Yemen)</td>
<td>Greek</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>Hebrew</td>
</tr>
<tr>
<td>Chinese (Hong Kong SAR)</td>
<td>Hungarian</td>
</tr>
<tr>
<td>Chinese (PRC)</td>
<td>Indonesian</td>
</tr>
<tr>
<td>Chinese (Singapore)</td>
<td>Italian (Italy)</td>
</tr>
<tr>
<td>Chinese (Taiwan)</td>
<td>Italian (Switzerland)</td>
</tr>
<tr>
<td>Croatian</td>
<td>Japanese</td>
</tr>
<tr>
<td>Czech</td>
<td>Korean</td>
</tr>
<tr>
<td>Danish (Denmark)</td>
<td>Latvian</td>
</tr>
<tr>
<td>Dutch (Belgium)</td>
<td>Norwegian (Bokmal)</td>
</tr>
<tr>
<td>Dutch (Netherlands)</td>
<td>Norwegian (Nynorsk)</td>
</tr>
<tr>
<td>English (Australia)</td>
<td>Polish</td>
</tr>
<tr>
<td>English (Belize)</td>
<td>Portuguese (Brazil)</td>
</tr>
<tr>
<td>English (Canada)</td>
<td>Portuguese (Portugal)</td>
</tr>
<tr>
<td>English (Ireland)</td>
<td>Romanian</td>
</tr>
<tr>
<td>English (New Zealand)</td>
<td>Russian</td>
</tr>
<tr>
<td>English (South Africa)</td>
<td>Serbian (Latin) (Yugoslavia)</td>
</tr>
<tr>
<td>English (United Kingdom)</td>
<td>Slovak</td>
</tr>
<tr>
<td>English (United States)</td>
<td>Slovenian</td>
</tr>
<tr>
<td>Estonian</td>
<td>Spanish (Mexico)</td>
</tr>
<tr>
<td>Farsi</td>
<td>Spanish (Spain)</td>
</tr>
<tr>
<td>Finnish</td>
<td>Swedish (Finland)</td>
</tr>
<tr>
<td>French (Canada)</td>
<td>Swedish (Sweden)</td>
</tr>
<tr>
<td>French (France)</td>
<td>Thai</td>
</tr>
<tr>
<td>French (Switzerland)</td>
<td>Turkish (Turkey)</td>
</tr>
<tr>
<td>German (Austria)</td>
<td>Ukrainian (Ukraine)</td>
</tr>
<tr>
<td>German (Germany)</td>
<td></td>
</tr>
</tbody>
</table>
Using the locale map

For Actuate Basic reports, the locale map specifies the following for the locales that Actuate supports:

- Formats for dates and times
- Page number formats
- Local and international currency symbols
- AM/PM symbols
- Plus, minus, and percent signs
- Decimal, grouping, date, time, and list separators
- Number of digits to group
- Number of digits after the decimal separator
- Input date mode
- Display of negative values
- Full and abbreviated month and day names
- Aggregation labels for the Actuate Query option

Understanding localemap.xml

The locale map resides in an XML file called localemap.xml. localemap.xml uses UTF-8 encoding. The portion of localemap.xml that specifies the default locale, the ANSI C locale, is shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SystemLocales name="Standard" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Header>
    <Copyright>
      Actuate Corporation
      Copyright(C) 2002-2010 Actuate Corporation. All rights reserved.
    </Copyright>
    <Version>
      <Name>RELEASE 1.0</Name>
      <Major>1</Major>
      <Minor>0</Minor>
    </Version>
  </Header>
</SystemLocales>
```

(continues)
<Locale ID="default">
  <DisplayName>ANSI C</DisplayName>
  <Code>en_US</Code>
  <FormatPatterns>
    <Date>
      <Short>M/d/yyyy</Short>
      <Medium>MMM d, yyyy</Medium>
      <Long>dddd, MMMM dd, yyyy</Long>
      <Full />
      <AB_Medium>dd-MMM-yy</AB_Medium>
    </Date>
    <Time>
      <Short>h:mm a</Short>
      <Medium>h:mm:ss a</Medium>
      <Long>h:mm:ss a</Long>
      <Full />
      <AB_Short>HH:mm</AB_Short>
      <AB_Medium>hh:mm a</AB_Medium>
    </Time>
    <DateTime>
      <Short>M/d/yyyy h:mm a</Short>
      <Medium>MMM d, yyyy h:mm:ss a</Medium>
      <Long>dddd, MMMM dd, yyyy h:mm:ss a</Long>
    </DateTime>
    <Timespan>
      <ShortMonth>M/yyyy</ShortMonth>
      <LongYear>yyyy</LongYear>
      <ShortWeek>
        <Pattern>'W'w yyyy</Pattern>
      </ShortWeek>
      <ShortQuarter>
        <Pattern>'Q'q yyyy</Pattern>
        <Quarter1>'Q1' yyyy</Quarter1>
        <Quarter2>'Q2' yyyy</Quarter2>
        <Quarter3>'Q3' yyyy</Quarter3>
        <Quarter4>'Q4' yyyy</Quarter4>
      </ShortQuarter>
      <ShortHalf>
        <Pattern>'H'l yyyy</Pattern>
        <Half1>'H1' yyyy</Half1>
        <Half2>'H2' yyyy</Half2>
      </ShortHalf>
    </Timespan>
  </FormatPatterns>
  <PageNumbers>
    <PageNOfM>
      <Long>Page $p of $c</Long>
      <Short>P $p/$c</Short>
    </PageNOfM>
  </PageNumbers>
</Locale>
Chapter 2, Formatting report data for multiple locales

(continues)
<table>
<thead>
<tr>
<th>Month</th>
<th>Short</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>April</td>
<td>Apr</td>
<td>April</td>
</tr>
<tr>
<td>May</td>
<td>May</td>
<td>May</td>
</tr>
<tr>
<td>June</td>
<td>Jun</td>
<td>June</td>
</tr>
<tr>
<td>July</td>
<td>Jul</td>
<td>July</td>
</tr>
<tr>
<td>August</td>
<td>Aug</td>
<td>August</td>
</tr>
<tr>
<td>September</td>
<td>Sep</td>
<td>September</td>
</tr>
<tr>
<td>October</td>
<td>Oct</td>
<td>October</td>
</tr>
<tr>
<td>November</td>
<td>Nov</td>
<td>November</td>
</tr>
<tr>
<td>December</td>
<td>Dec</td>
<td>December</td>
</tr>
</tbody>
</table>
<AverageHeading>
  <Short>Ave.</Short>  
  <Long>Average</Long>
</AverageHeading>

<AverageLabel>
  <Short>Ave:</Short>  
  <Long>Average:</Long>
</AverageLabel>

<SumHeading>
  <Short>Sum</Short>  
  <Long>Sum</Long>
</SumHeading>

<SumLabel>
  <Short>Sum:</Short>  
  <Long>Sum:</Long>
</SumLabel>

<CountHeading>
  <Short>Count</Short>  
  <Long>Count</Long>
</CountHeading>

<CountLabel>
  <Short>Count:</Short>  
  <Long>Count:</Long>
</CountLabel>

<MaxHeading>
  <Short>Max.</Short>  
  <Long>Maximum</Long>
</MaxHeading>

<MaxLabel>
  <Short>Max:</Short>  
  <Long>Maximum:</Long>
</MaxLabel>

<MinHeading>
  <Short>Min.</Short>  
  <Long>Minimum</Long>
</MinHeading>

<MinLabel>
  <Short>Min:</Short>  
  <Long>Minimum:</Long>
</MinLabel>

<OverallHeading>
  <Short>Overall</Short>  
  <Long>Overall</Long>
</OverallHeading>

<OverallLabel>

(continues)
Table 2-1 lists the XML tags in localemap.xml with a description of each tag.

<table>
<thead>
<tr>
<th>XML tag</th>
<th>Child tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregationLabels</td>
<td></td>
<td>Full and abbreviated headings and aggregation labels for the Actuate Query option, including: Average, Count, Maximum, Minimum, Overall, Sum</td>
</tr>
<tr>
<td>Code</td>
<td></td>
<td>The ISO 639 language code plus the ISO 3166 country code</td>
</tr>
<tr>
<td>CurrencySymbol</td>
<td>Positive</td>
<td>Position of currency symbol for positive values: PrecedeNoSpace, PrecedeWithSpace, SucceedNoSpace, SucceedWithSpace</td>
</tr>
<tr>
<td></td>
<td>Negative</td>
<td>Position of currency symbol for negative values: PrecedeNoSpace, PrecedeWithSpace, SucceedNoSpace, SucceedWithSpace</td>
</tr>
<tr>
<td>Date</td>
<td>Short</td>
<td>Short date format</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
<td>Medium date format used by Actuate Information Console and Management Console</td>
</tr>
<tr>
<td></td>
<td>Long</td>
<td>Long date format</td>
</tr>
</tbody>
</table>
### Table 2-1  XML tags in localemap.xml (continued)

<table>
<thead>
<tr>
<th>XML tag</th>
<th>Child tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date (continued)</td>
<td>Full</td>
<td>Full date format</td>
</tr>
<tr>
<td></td>
<td>AB_Medium</td>
<td>Medium date format provided for backward compatibility</td>
</tr>
<tr>
<td>DateTime</td>
<td>Short</td>
<td>Short date/time format</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
<td>Medium date/time format</td>
</tr>
<tr>
<td></td>
<td>Long</td>
<td>Long date/time format</td>
</tr>
<tr>
<td>DaysOfWeek</td>
<td>Short</td>
<td>Abbreviated day names</td>
</tr>
<tr>
<td></td>
<td>Full</td>
<td>Full day names</td>
</tr>
<tr>
<td>DisplayName</td>
<td></td>
<td>Locale name displayed on the Information Console, and Management Console login pages</td>
</tr>
<tr>
<td>Locale ID</td>
<td></td>
<td>For every locale except the default locale, the ISO 639 language code plus the ISO 3166 country code. For the default locale, the Locale ID is &quot;default&quot;.</td>
</tr>
<tr>
<td>MonthsOfYear</td>
<td>Short</td>
<td>Abbreviated month names</td>
</tr>
<tr>
<td></td>
<td>Full</td>
<td>Full month names</td>
</tr>
<tr>
<td>PageNumbers</td>
<td>Long</td>
<td>Long page number, for example Page 19 of 42</td>
</tr>
<tr>
<td></td>
<td>Short</td>
<td>Short page number, for example P 19/42</td>
</tr>
<tr>
<td></td>
<td>PageN</td>
<td>Page number, for example Page 19</td>
</tr>
<tr>
<td>Positions</td>
<td>Grouping</td>
<td>Number of digits to group for numbers</td>
</tr>
<tr>
<td></td>
<td>CurrencyGrouping</td>
<td>Number of digits to group for currency</td>
</tr>
<tr>
<td></td>
<td>FractionDigits</td>
<td>Number of digits after the decimal separator for currency</td>
</tr>
<tr>
<td></td>
<td>IntFractionDigits</td>
<td>Number of digits after the decimal separator for numbers</td>
</tr>
<tr>
<td>InputDateMode</td>
<td></td>
<td>Specifies the order in which the day, month, and year are entered:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = month, day, year</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = day, month, year</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = year, month, day</td>
</tr>
<tr>
<td>Separators</td>
<td>Decimal</td>
<td>Decimal separator for numbers</td>
</tr>
<tr>
<td></td>
<td>Grouping</td>
<td>Grouping separator for numbers</td>
</tr>
<tr>
<td></td>
<td>CurrencyDecimal</td>
<td>Decimal separator for currency</td>
</tr>
</tbody>
</table>

(continues)
<table>
<thead>
<tr>
<th>XML tag</th>
<th>Child tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separators</td>
<td>CurrencyGrouping</td>
<td>Grouping separator for currency</td>
</tr>
<tr>
<td></td>
<td>Date</td>
<td>Date separator</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>Time separator</td>
</tr>
<tr>
<td></td>
<td>List</td>
<td>List separator used when an Information Console user downloads search results as comma-delimited data</td>
</tr>
<tr>
<td>Sign</td>
<td>Positive</td>
<td>Position of sign for positive values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parentheses = no sign, enclose number in parentheses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Precede</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PrecedeCurrencySymbol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Succeed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SucceedWithSpace</td>
</tr>
<tr>
<td></td>
<td>Negative</td>
<td>Position of sign for negative values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parentheses = no sign, enclose number in parentheses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Precede</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PrecedeCurrencySymbol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Succeed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SucceedWithSpace</td>
</tr>
<tr>
<td>Symbols</td>
<td>Currency</td>
<td>Local currency symbol</td>
</tr>
<tr>
<td></td>
<td>IntCurrency</td>
<td>International currency symbol</td>
</tr>
<tr>
<td></td>
<td>Plus</td>
<td>Plus sign</td>
</tr>
<tr>
<td>Symbols</td>
<td>Minus</td>
<td>Minus sign</td>
</tr>
<tr>
<td></td>
<td>AM</td>
<td>AM symbol</td>
</tr>
<tr>
<td></td>
<td>PM</td>
<td>PM symbol</td>
</tr>
<tr>
<td></td>
<td>Infinity</td>
<td>Infinity symbol</td>
</tr>
<tr>
<td></td>
<td>NotANumber</td>
<td>Not a number symbol</td>
</tr>
<tr>
<td></td>
<td>Percent</td>
<td>Percent sign</td>
</tr>
<tr>
<td>Time</td>
<td>Short</td>
<td>Short time format used by Information Console and Management Console</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
<td>Medium time format used by Information Console and Management Console</td>
</tr>
</tbody>
</table>

Table 2-1 XML tags in localemap.xml (continued)
Table 2-1  XML tags in localemap.xml (continued)

<table>
<thead>
<tr>
<th>XML tag</th>
<th>Child tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (continued)</td>
<td>Long</td>
<td>Long time format</td>
</tr>
<tr>
<td></td>
<td>Full</td>
<td>Full time format</td>
</tr>
<tr>
<td></td>
<td>AB_Short</td>
<td>Short time format provided for backward compatibility</td>
</tr>
<tr>
<td></td>
<td>AB_Medium</td>
<td>Medium time format provided for backward compatibility</td>
</tr>
<tr>
<td>Timespan</td>
<td>LongYear</td>
<td>Year, for example 2003</td>
</tr>
<tr>
<td></td>
<td>ShortMonth</td>
<td>Month of year, for example 10/2003</td>
</tr>
<tr>
<td></td>
<td>ShortWeek</td>
<td>Week of year includes both a format pattern and named time period, for example W39 2003</td>
</tr>
<tr>
<td></td>
<td>ShortQuarter</td>
<td>Quarter of year includes both a format pattern and named time period, for example Q4 2003</td>
</tr>
<tr>
<td></td>
<td>ShortHalf</td>
<td>Half of year includes both a format pattern and named time period, for example H2 2003</td>
</tr>
</tbody>
</table>

Table 2-2 lists the symbols that you can use to construct date format patterns with a description of each symbol.

Table 2-2  Symbols for date format patterns

<table>
<thead>
<tr>
<th>Date symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Era designator</td>
</tr>
<tr>
<td>y</td>
<td>Year</td>
</tr>
<tr>
<td>M</td>
<td>Month</td>
</tr>
<tr>
<td>d</td>
<td>Day of month</td>
</tr>
<tr>
<td>E</td>
<td>Weekday name</td>
</tr>
<tr>
<td>D</td>
<td>Day of year</td>
</tr>
<tr>
<td>F</td>
<td>Day of week in month, for example 2 represents the second Wednesday in July</td>
</tr>
<tr>
<td>w</td>
<td>Week of year</td>
</tr>
<tr>
<td>W</td>
<td>Week of month</td>
</tr>
<tr>
<td>q</td>
<td>Quarter of year</td>
</tr>
<tr>
<td>l (the letter el)</td>
<td>Half of year</td>
</tr>
<tr>
<td>'</td>
<td>Escape character</td>
</tr>
<tr>
<td>&quot;</td>
<td>(Two single quotes) Single quote</td>
</tr>
</tbody>
</table>
Table 2-3 lists the symbols that can be used to construct time format patterns with a description of each symbol.

**Table 2-3**  Symbols for time format patterns

<table>
<thead>
<tr>
<th>Time symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Hour (1-12)</td>
</tr>
<tr>
<td>H</td>
<td>Hour (0-23)</td>
</tr>
<tr>
<td>m</td>
<td>Minute</td>
</tr>
<tr>
<td>s</td>
<td>Second</td>
</tr>
<tr>
<td>S</td>
<td>Millisecond</td>
</tr>
<tr>
<td>a</td>
<td>AM/PM symbol</td>
</tr>
<tr>
<td>k</td>
<td>Hour of day (1-24)</td>
</tr>
<tr>
<td>K</td>
<td>Hour of AM/PM (0-11)</td>
</tr>
<tr>
<td>z</td>
<td>Time zone</td>
</tr>
<tr>
<td>`</td>
<td>Escape character</td>
</tr>
<tr>
<td>&quot;</td>
<td>(Two single quotes) Single quote</td>
</tr>
</tbody>
</table>

Table 2-4 lists several format patterns and sample results for the US English locale.

**Table 2-4**  Sample format patterns

<table>
<thead>
<tr>
<th>Format pattern</th>
<th>Sample result</th>
</tr>
</thead>
<tbody>
<tr>
<td>yyyy.MM.dd G 'at' HH:mm:ss z</td>
<td>1996.07.01 AD at 15:08:56 PDT</td>
</tr>
<tr>
<td>EEE, MMM d, &quot;yy</td>
<td>Wed, Jul 1, ‘96</td>
</tr>
<tr>
<td>h:mm a</td>
<td>1:08 PM</td>
</tr>
<tr>
<td>hh 'o’clock' a, zzzz</td>
<td>12 o’clock PM, Pacific Daylight Time</td>
</tr>
<tr>
<td>K:mm a, z</td>
<td>0:00 PM, PST</td>
</tr>
<tr>
<td>yyyy.MMMMMMMdd GGG hh:mm aaa</td>
<td>1996.July:01 AD 02:08 PM</td>
</tr>
</tbody>
</table>

**About the AB tags**

The following tags are provided for backward compatibility with Actuate 5:

- **AB_Medium** in the Date section
- **AB_Short** in the Time section
- **AB_Medium** in the Time section
For example, if you use the Medium Date keyword to format a date/time control in an Actuate Basic report design, the date displays with the format pattern specified by the AB_Medium tag in the Date section, dd-MMM-yy. The date does not display with the format pattern specified by the Medium tag in the Date section, MMM d, yyyy. The format pattern specified by the AB_Medium tag is the Actuate Basic Medium Date format pattern used in Actuate 5. The format pattern specified by the Medium tag is used by Actuate Information Console and Management Console.

**Using localemap.xml**

The following products use localemap.xml:

- iServer
- Actuate web-based products
- e.Report Designer Professional

localemap.xml must be consistent across the iServer System. For example, if the system administrator makes a change to the localemap.xml file used by an Actuate server, the change must be applied to the localemap.xml files used by the web-based products and e.Report Designer Professional as well.

**Using localemap.xml with Actuate iServer**

An Actuate server’s localemap.xml and the locale determine the formats used in an Actuate Basic report. localemap.xml is located in $AC_SERVER_HOME/etc. If localemap.xml is modified, the system administrator must restart every Actuate server in the iServer System. Every Actuate server in the iServer System must use the same localemap.xml.

Because locale processing is handled by localemap.xml, it does not matter which locale an Actuate server runs in.

**Using localemap.xml with Actuate web-based products**

The following web-based products use localemap.xml:

- Information Console
  - Information Console’s localemap.xml and the locale determine the formats used in the Information Console user interface, for example the date/time format in the Finished column on the My Jobs - Completed page.
  - localemap.xml is located in Program Files\Actuate11\iServer \servletcontainer\iportal\WEB-INF on Windows XP platforms and $AC_SERVER_HOME/servletcontainer/iportal/WEB-INF on UNIX platforms.

- Actuate Management Console
Management Console’s localemap.xml and the locale determine the formats used in the Management Console user interface, for example the date/time format in the Finished column on the Jobs - Completed page. localemap.xml is located in \Program Files\Actuate11\iServer\servletcontainer\mgmtconsole \WEB-INF on Windows XP platforms and $AC_SERVER_HOME /servletcontainer/mgmtconsole/WEB-INF on UNIX platforms.

If localemap.xml is modified, the web server or the web application must be restarted.

Using localemap.xml with e.Report Designer Professional

e.Report Designer Professional’s localemap.xml and the locale determine the formats used in an Actuate Basic report on the desktop. The locale is determined as described in “Running reports with Actuate e.Report Designer Professional,” earlier in this chapter. localemap.xml is located in \Program Files\Actuate11 \Config on Windows XP platforms. If localemap.xml is modified, e.Report Designer Professional must be restarted.

Creating a custom locale

The system administrator can change the settings for the locales that Actuate supports or create a custom locale by modifying localemap.xml. You must use a text editor that supports UTF-8 encoding. The custom locale ID must be an alphanumeric ASCII string of fewer than 16 characters. Do not use non-alphanumeric characters such as comma and space.

Use Java date/time formats. Do not use Windows date/time formats.

Do not use time formats that do not indicate whether the time is AM or PM. For example, do not use the time format "hh:mm". Instead, use "hh:mm a" or "HH:mm". If you use the time format "hh:mm", the time is displayed as 08:55. If you use the time format "hh:mm a", the time is displayed as 08:55 PM. If you use the time format "HH:mm", the time is displayed as 20:55.

If the custom locale does not set an attribute, the attribute setting for the default locale in localemap.xml is used. For example, if the custom locale does not set the long date format, the long date format for the default locale is used.

Specifying a locale with Actuate Information Console

When a user logs in to Actuate Information Console, they can choose a locale from the drop-down list on the login page. This list contains the locales defined in localemap.xml. The locale chosen by the user is included in Information Console URLs and in the Actuate SOAP API request that Information Console sends to the iServer System. The iServer System then formats reports for this locale.
For example:

- The user chooses the locale French (France) on the Information Console login page, as shown in Figure 2-2.

![Figure 2-2 Choosing a locale on the Information Console login page](image)

- The user selects a report to view, yielding the following URL:


- Information Console sends an Actuate SOAP API request to the iServer System. The locale is included in the header. The request uses UTF-8 encoding:

  ```xml
  <ENC:Header>
    <Authid>??</Authid>
    <TargetVolume>sales</TargetVolume>
    <Locale>fr_FR</Locale>
  </ENC:Header>
  ```

- The iServer System formats the report for the French (France) locale.

- The DHTML report appears in the user’s web browser.

### Setting a report’s locale

The report developer can set a report’s locale with the top-level report component’s Locale property, or by overriding methods. The Locale setting determines the formats for dates, times, currency, and numbers.

### Setting the Locale property

To set the Locale property, choose a locale from the drop-down in the top-level report component’s Component Editor, as shown in Figure 2-3.
Figure 2-3  Setting the Locale property for a report

Setting different locales for generating, viewing, and printing a report

The report developer can set different locales for generating, viewing, and printing a report by overriding the following methods:

- Function GetFactoryLocale( defaultLocale As String ) As String
- Function GetViewLocale( defaultLocale As String ) As String
- Function GetPrintLocale( defaultLocale As String ) As String

These methods are associated with the top-level report component.

Using GetFactoryLocale( )

To set a locale for generating a report, override GetFactoryLocale( ). GetFactoryLocale( ) is called before the report is generated. If this method is not overridden, the value of the Locale property is used. For example:

Function GetFactoryLocale( defaultLocale As String ) As String
    GetFactoryLocale = Super::GetFactoryLocale( defaultLocale )
    GetFactoryLocale = LocaleParam
End Function

where LocaleParam is a parameter of type String. The return value of GetFactoryLocale( ) is stored in the ROI.
Using GetViewLocale( )

To set a locale for viewing a report, override GetViewLocale( ). GetViewLocale( ) is called before the report is viewed. If this method is not overridden, the Locale value stored in the ROI is used. For example:

```vbscript
Function GetViewLocale( defaultLocale As String ) As String
    GetViewLocale = Super::GetViewLocale( defaultLocale )
    GetViewLocale = LocaleParam
End Function
```

where LocaleParam is a parameter of type String. The return value of GetViewLocale( ) is not stored in the ROI; it is used only for viewing the report.

Using GetPrintLocale( )

To set a locale for printing a report on an Actuate server, override GetPrintLocale( ). GetPrintLocale( ) is called before the report is printed on the Actuate server. If this method is not overridden, the Locale value stored in the ROI is used. For example:

```vbscript
Function GetPrintLocale( defaultLocale As String ) As String
    GetPrintLocale = Super::GetPrintLocale( defaultLocale )
    GetPrintLocale = LocaleParam
End Function
```

where LocaleParam is a parameter of type String. The return value of GetPrintLocale( ) is not stored in the ROI; it is used only for printing the report on the Actuate server. On the desktop, the printing locale is the same as the viewing locale.

Setting different locales for generating and viewing a report

1. Create two parameters of type String:
   - generatingLocaleParam
   - viewingLocaleParam

2. Override GetFactoryLocale( ):

   ```vbscript
   Function GetFactoryLocale( defaultLocale As String ) As String
       GetFactoryLocale = Super::GetFactoryLocale( defaultLocale )
       If generatingLocaleParam <> "" Then
           GetFactoryLocale = generatingLocaleParam
       End If
   End Function
   ```

3. Create a String variable called viewingLocale on the top-level report component.

Chapter 2, Formatting report data for multiple locales  25
Override the top-level report component’s Start( ) method:

```vbnet
Sub Start( )
    Super::Start( )
    If viewingLocaleParam <> "" Then
        viewingLocale = viewingLocaleParam
    End If
End Sub
```

Override GetViewLocale( ):

```vbnet
Function GetViewLocale( defaultLocale As String ) As String
    GetViewLocale = Super::GetViewLocale( defaultLocale )
    If viewingLocale <> "" Then
        GetViewLocale = viewingLocale
    End If
End Function
```

---

**Parsing strings with Actuate Basic functions**

Many Actuate Basic functions parse strings. Some of these functions parse strings without taking the report’s runtime locale into account, while others parse strings according to the rules of the runtime locale.

For more information about Actuate Basic functions, see *Programming with Actuate Basic*.

**Using locale-independent parsing**

The ParseNumeric and ParseDate functions parse strings without taking the report's runtime locale into account.

**Using the ParseNumeric function**

To parse a numeric expression without taking the report's runtime locale into account, use the ParseNumeric function. ParseNumeric takes a numeric expression of type String and the decimal separator, thousands separator, and currency symbol used in the expression and returns a Double. For example:

- `ParseNumeric("123,456.78", ".", ",", NULL) returns 123456.78`
- `ParseNumeric("123,456.78", ",", ".", NULL) returns 123456.78`
- `ParseNumeric("123!456*78", ",", ",", NULL) returns 123456.78`
- `ParseNumeric("$1,500.00", ",", ",", "$") returns 1500.00`

If the decimal separator or thousands separator evaluates to Null or empty string, the decimal separator or thousands separator specified by the runtime locale is used.
Using the ParseDate function

To parse a date expression for a specific locale, use the ParseDate function. ParseDate takes a date expression of type String, the date expression’s format, and the locale code and returns a Date. For example, to parse a date expression for the French (France) locale:

```vbscript
Dim d_date As Date
d_date = ParseDate("25/12/01", "dmy", "fr_FR")
```

If the locale code evaluates to Null or is not valid, the report’s runtime locale is used.

Table 2-5 lists example formats and date expressions.

<table>
<thead>
<tr>
<th>Format</th>
<th>Date expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ymd</td>
<td>1999-03-04</td>
<td>Year-month-day with four-digit year.</td>
</tr>
<tr>
<td>mdy</td>
<td>Nov. 12, 1972</td>
<td>Month-day-year with four-digit year.</td>
</tr>
<tr>
<td>mdy19</td>
<td>11/12/72</td>
<td>Month-day-year with two-digit year assumed to be in the twentieth century.</td>
</tr>
<tr>
<td>dmyp</td>
<td>28-3-02</td>
<td>Day-month-year with two-digit year that uses the default Actuate pivot date.</td>
</tr>
<tr>
<td>mdy19p25</td>
<td>4/1/83</td>
<td>Month-day-year with century and pivot. Use this format when importing data from a file.</td>
</tr>
<tr>
<td>ymdt</td>
<td>1999-01-28 17:15</td>
<td>Year-month-day with 24-hour time.</td>
</tr>
<tr>
<td>mdy12t</td>
<td>1/4/94 9:12 AM</td>
<td>Month-day-year with 12-hour time.</td>
</tr>
<tr>
<td>w?mdy12t?</td>
<td>Sat. May 14, 1999 12:43 PM</td>
<td>Month-day-year with optional weekday and time.</td>
</tr>
</tbody>
</table>

Understanding locale-dependent parsing

Many Actuate Basic functions parse strings according to the rules of the report’s runtime locale:

- Understanding functions that operate on numeric expressions
- Understanding functions that operate on date expressions

Locale-dependent parsing is not recommended.
Understanding functions that operate on numeric expressions

Several Actuate Basic functions operate on numeric expressions of type String. These functions parse the numeric expression according to the rules of the report's runtime locale, and are described in Table 2-6. For example:

\[
\text{CDbl("123,456") returns 123456.00 for the US locale.}
\]
\[
\text{CDbl("123,456") returns 123.456 for the French (France) locale.}
\]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
<td>Returns the absolute value for a number or expression</td>
</tr>
<tr>
<td>Atn</td>
<td>Gives the arctangent of a number</td>
</tr>
<tr>
<td>CCur</td>
<td>Converts a numeric expression to the Currency data type</td>
</tr>
<tr>
<td>CDbl</td>
<td>Converts a numeric expression to the Double data type</td>
</tr>
<tr>
<td>CInt</td>
<td>Converts a numeric expression to the Integer data type</td>
</tr>
<tr>
<td>CLng</td>
<td>Converts a numeric expression to the Long data type</td>
</tr>
<tr>
<td>Cos</td>
<td>Returns the cosine of an angle</td>
</tr>
<tr>
<td>CSng</td>
<td>Converts a numeric expression to the Single data type</td>
</tr>
<tr>
<td>CStr</td>
<td>Converts an expression to the String data type</td>
</tr>
<tr>
<td>Exp</td>
<td>Raises e to the specified power</td>
</tr>
<tr>
<td>Fix</td>
<td>Removes the fractional part of a numeric expression and returns whatever integer remains</td>
</tr>
<tr>
<td>Int</td>
<td>Returns the largest integer that is less than or equal to a given numeric expression</td>
</tr>
<tr>
<td>IsNumeric</td>
<td>Tests whether the type of a variable is or can be converted to Integer, Long, Single, Double, or Currency</td>
</tr>
<tr>
<td>Oct, Oct$</td>
<td>Converts a numeric expression from decimal to octal notation, and from numeric to string</td>
</tr>
<tr>
<td>Sin</td>
<td>Gives the sine of an angle</td>
</tr>
<tr>
<td>Sqr</td>
<td>Gives the square root of a number</td>
</tr>
<tr>
<td>Str, Str$</td>
<td>Converts a numeric expression to a String</td>
</tr>
<tr>
<td>Tan</td>
<td>Returns the tangent of an angle</td>
</tr>
</tbody>
</table>
Understanding functions that operate on date expressions

Several Actuate Basic functions operate on date expressions of type String. These functions parse the date expression according to the rules of the report’s runtime locale, and are described in Table 2-7.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDate</td>
<td>Converts an expression to a Date</td>
</tr>
<tr>
<td>CVDate</td>
<td>Converts an expression to a Variant of VarType 7 (Date)</td>
</tr>
<tr>
<td>DateAdd</td>
<td>Returns a date to which a specified time interval has been added</td>
</tr>
<tr>
<td>DateDiff</td>
<td>Calculates and returns the time difference between two specified dates</td>
</tr>
<tr>
<td>DatePart</td>
<td>Returns a specified component of a given date</td>
</tr>
<tr>
<td>DateValue</td>
<td>Returns a date variant that represents the date of the supplied string</td>
</tr>
<tr>
<td>Day</td>
<td>Returns an integer between 1 and 31, inclusive, that represents the day of the month for a supplied date argument</td>
</tr>
<tr>
<td>Hour</td>
<td>Returns the hour of the day as an integer from 0 (midnight) to 23 (11:00 p.m.), inclusive, based on a supplied date expression</td>
</tr>
<tr>
<td>IsDate</td>
<td>Determines whether the given argument can be converted to a date</td>
</tr>
<tr>
<td>Minute</td>
<td>Returns an integer from 0 to 59, inclusive, that represents the minute of the hour specified by a supplied date expression</td>
</tr>
<tr>
<td>Month</td>
<td>Returns an integer between 1 and 12, inclusive, that represents the month of the year for a supplied date argument</td>
</tr>
<tr>
<td>Second</td>
<td>Returns an integer from 0 to 59, inclusive, that represents the second of the minute specified by a supplied date expression</td>
</tr>
<tr>
<td>TimeValue</td>
<td>Returns a Date variant representing a time of day, based on a supplied string</td>
</tr>
<tr>
<td>Weekday</td>
<td>Returns an integer between 1 (for Sunday) and 7 (for Saturday) that represents the day of the week for a supplied date argument</td>
</tr>
<tr>
<td>Year</td>
<td>Returns an integer between 100 and 9999, inclusive, that represents the year of a supplied date argument</td>
</tr>
</tbody>
</table>
Formatting dates, times, currency, and numbers

Use the Actuate Basic functions Format and Format$ to format dates, times, currency, and numbers for a specific locale.

For more information about Actuate Basic functions, see Programming with Actuate Basic.

Formatting dates and times

To format a date or time for a specific locale, use the Format or Format$ function. Format[$] takes a date variant, a format keyword or string, and the locale code. For example, to format a date variant for the French (France) locale:

    Format$(DateVar, "Long date", "fr_FR")

If the locale code evaluates to Null or is not valid, the report’s runtime locale is used.

The following format keywords are locale-dependent. These format keywords are defined in localemap.xml, not in the Windows Regional Settings Properties:

- General date
- Long date
- Medium date
- Short date
- Long time
- Medium time
- Short time
- Week
- Month
- Quarter
- Half
- Year

The following format strings are locale-dependent:

- ddd
  Three-letter abbreviation for day of the week specified in localemap.xml.
- dddd
  Full name of day of the week specified in localemap.xml.
■ mmm
   Three-letter abbreviation for month name specified in localemap.xml.

■ mmmm
   Full name of the month specified in localemap.xml.

■ @
   Date separator specified in localemap.xml, for example mm@dd@yyyy.

■ AMPM
   AM/PM symbols specified in localemap.xml.

### Formatting currency and numbers

To format a number for a specific locale, use the Format or Format$ function. Format$ takes a numeric expression, a format keyword or string, and the locale code. For example, to format a numeric expression for the US English locale:

```
Format(3434.2899, "Currency", "en_US")
```

You can enter the numeric expression in either of two ways:

- Enter the numeric expression using the C locale.
  Do not use the thousands separator, for example 3434.2899.

- Enter the numeric expression using the report’s runtime locale and enclose it in quotation marks ("').
  For example, if the report’s runtime locale will be French (France), enter "3 434,2899".

If the locale code evaluates to Null or is not valid, the report’s runtime locale is used to format the numeric expression.

If you use a format keyword such as General number, the thousands separator and the decimal separator in the formatted result are dependent on the specified locale. If you use the Currency format keyword, the currency symbol in the formatted result is dependent on the specified locale. If you use the Percent format keyword, the percent sign in the formatted result is dependent on the specified locale.

If you use a format string such as ($) #,##0.00:

- The ($) is replaced with the currency symbol for the specified locale.
- The comma (,) is replaced with the thousands separator for the specified locale.
- The period (.) is replaced with the decimal separator for the specified locale.

If you use a format string such as ###%, the % is replaced with the percent sign for the specified locale.
Using a pre-Euro currency symbol

Several European locales use the Euro currency symbol. If you want to use the pre-Euro currency symbol for a locale, you must modify the Currency and IntCurrency attributes for the locale in localemap.xml. For example, if you want to use the French franc currency symbol for the French (France) locale, modify the Currency and IntCurrency attributes as follows:

```xml
<Currency>F</Currency>
<IntCurrency>FRF</IntCurrency>
```

You must use a text editor that supports UTF-8 encoding.

Getting the locale name and locale attributes

Use the Actuate Basic functions GetLocaleName() and GetLocaleAttribute() to get the locale name and locale attributes.

For more information about Actuate Basic functions, see Programming with Actuate Basic.

Understanding GetLocaleName()

GetLocaleName() returns the name of the runtime locale. GetLocaleName() returns a string.

Understanding GetLocaleAttribute()

GetLocaleAttribute() returns an attribute for the specified locale. GetLocaleAttribute() returns a string. For example, to get the currency symbol for the French (France) locale:

```
GetLocaleAttribute("fr_FR", AC_LOCALE_CURRENCY)
```

Table 2-8 lists the locale attributes that can be returned by GetLocaleAttribute().

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC_LOCALE_CURRENCY</td>
<td>Currency symbol</td>
</tr>
<tr>
<td>AC_LOCALE_CURRENCY_FORMAT</td>
<td>Format for currency</td>
</tr>
<tr>
<td>AC_LOCALE_CURRENCY_RADIX</td>
<td>Decimal separator for currency</td>
</tr>
<tr>
<td>AC_LOCALE_CURRENCY_THOUSAND_SEPARATOR</td>
<td>Thousands separator for currency</td>
</tr>
<tr>
<td>AC_LOCALE_DATE_LONG</td>
<td>Long date format</td>
</tr>
</tbody>
</table>
Chapter 2, Formatting report data for multiple locales

Table 2-8  Locale attributes that GetLocaleAttribute( ) can return

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC_LOCALE_DATE_SEPARATOR</td>
<td>Date separator</td>
</tr>
<tr>
<td>AC_LOCALE_DATE_SHORT</td>
<td>Short date format</td>
</tr>
<tr>
<td>AC_LOCALE_MONTHS_LONG</td>
<td>Comma-separated list of month names</td>
</tr>
<tr>
<td>AC_LOCALE_MONTHS_SHORT</td>
<td>Comma-separated list of month abbreviations</td>
</tr>
<tr>
<td>AC_LOCALE_NUM_RADIX</td>
<td>Decimal separator for numbers</td>
</tr>
<tr>
<td>AC_LOCALE_NUM_THOUSAND_SEPARATOR</td>
<td>Thousands separator for numbers</td>
</tr>
<tr>
<td>AC_LOCALE_TIME_AM_STRING</td>
<td>AM string</td>
</tr>
<tr>
<td>AC_LOCALE_TIME_FORMAT</td>
<td>Time format</td>
</tr>
<tr>
<td>AC_LOCALE_TIME_PM_STRING</td>
<td>PM string</td>
</tr>
<tr>
<td>AC_LOCALE_TIME_SEPARATOR</td>
<td>Time separator</td>
</tr>
<tr>
<td>AC_LOCALE_WEEKDAYS_LONG</td>
<td>Comma-separated list of day names</td>
</tr>
<tr>
<td>AC_LOCALE_WEEKDAYS_SHORT</td>
<td>Comma-separated list of day abbrevations</td>
</tr>
</tbody>
</table>

Using GetLocaleName( ) and GetLocaleAttribute( )

The following examples show how to use GetLocaleName( ) and GetLocaleAttribute( ).

Parse the numeric string 1.001,99:

```
sep1000 = GetLocaleAttribute("fr_FR", AC_LOCALE_NUM_THOUSAND_SEPARATOR)
radix = GetLocaleAttribute("fr_FR", AC_LOCALE_NUM_RADIX)
ParseNumeric("1.001,99", sep1000, radix)
```

Get a comma-separated list of day abbreviations for the runtime locale:

```
dayShortNames = GetLocaleAttribute(GetLocaleName( ), AC_LOCALE_WEEKDAYS_SHORT)
```

If the runtime locale is US English, the string dayShortNames contains:

```
Mon, Tue, Wed, Thu, Fri, Sat, Sun
```
Understanding parameter handling

When a user runs a report, Actuate converts report parameter values to a locale-independent format for internal processing. It is not necessary for the report user or the report developer to know the locale in which the report will run. When a report developer creates a report parameter, he enters the parameter’s default value using the conventions of the locale in which he is running e.Report Designer Professional. When a user runs a report, he enters the parameter’s value using the conventions of the locale he chose when he logged in to Actuate Information Console. For information about ad hoc parameters and QBE syntax, see Using Information Console.

The report developer must save the report design (.rod), open the report design, and generate the report executable (.rox) using the same locale. This rule also applies to component libraries (.rol).

Example

A report developer is running e.Report Designer Professional in the US English locale. He creates a parameter called Start date and enters the default value 10/25/2002. When he runs the report, the default value for Start date appears as 10/25/2002 in the Requester dialog. He enters the value 11/30/2002. In the report, Start date is formatted according to the conventions of the runtime locale. If the runtime locale is US English, Start date appears in the report as 11/30/2002. If the runtime locale is French (France), Start date appears in the report as 30/11/2002.

A user logs in to Information Console and chooses the French (France) locale. When the user runs the report, the default value for Start date appears as 25/10/2002 on the Parameters page. He enters the value 30/11/2002. In the report, Start date is formatted according to the conventions of the runtime locale. If the runtime locale is French (France), Start date appears in the report as 30/11/2002. If the runtime locale is US English, Start date appears in the report as 11/30/2002.

Designing Japanese reports

A Japanese edition of e.Report Designer Professional is available. e.Report Designer Professional uses the Japanese resource files if:

- The operating system locale is Japanese.
- The Japanese executable, <RDPro_HOME>nls\jpn\erdpro.jpn, exists.
The Japanese resource files provide
- A Japanese user interface.
- Japanese messages.
- Japanese format patterns, such as imperial year ("gg") and weekday name ("aaa").
- Default font mapping to the Japanese fonts MS Gothic and MS Mincho.

Using a localized sfdata database and externalized strings

You can create a localized version of the Detail report using a localized sfdata database and a text file that contains the strings used in the report. This section uses an example to demonstrate the process. Figure 2-4 shows an example of a localized report.

![Example of a localized report](image)

About the localized sfdata database

This version of the Detail report connects to a French localized version of the sfdata database. For example, the first five rows of the items table are shown in Figure 2-5.

![Example of a localized database](image)
About the externalized strings

The strings in the French Detail report are externalized in a text file. The text file is a character separated value file with two fields in each line:

- The first field is the label control’s fully qualified component name in the report design.
- The second field is the appropriate French string.

The file contains one line for each label control component used in the report design:

- `ReportTitle::LabelControl, 1999 T2 Prévisions des ventes pour la région Est`
- `ReportTitle::LabelControl1, MultiChip Technology`
- `ReportTitle::LabelControl2, Total des prévisions des ventes :`
- `OfficeTitleFrame::LabelControl1, Total des prévisions des ventes :
...

About the hash array library

The report design uses a hash array library to associate the correct French string with a label control component. The hash array library is in a separate BAS file. The BAS file declares a global variable, labelDictionary, and a class, MyDictionary. The variable labelDictionary can hold a reference to an instance of the class MyDictionary. All components in the report design, including those in libraries, have access to labelDictionary:

```bas
Declare
    Global labelDictionary As MyDictionary
End Declare

Class MyDictionary
    Dim KeyLen(9) As Integer
    Dim Key0( ) As String
    Dim Key1( ) As String
    ...  
End Class
```

The following methods are associated with the class MyDictionary:

- `Sub New()`
- `Sub Delete()`
- `Function ComputeHashKey(strKey as String)`
- `Sub Add(strKey as String, strValue as String)`
- `Function GetValue(strKey as String)`
Processing the text file

The top-level report component’s Start() method processes the text file label.txt:

```vbscript
Sub Start()
    Super::Start()
    ' Insert your code here
    Dim fileNumber As Integer
    Dim labelName As String
    Dim labelValue As String
    Set labelDictionary = New MyDictionary
    fileNumber = FreeFile
    Open "label.txt" For Input As #fileNumber
    Do While Not EOF(fileNumber)
        Input #fileNumber, labelName, labelValue
        labelDictionary.Add(labelName, labelValue)
    Loop
    Close #fileNumber
End Sub
```

The Start() method:

- Initializes the global variable labelDictionary as a reference to an instance of the class MyDictionary.
- Opens the text file label.txt.
- Calls labelDictionary’s Add() method to place the contents of label.txt in a hash array.
- Closes label.txt.

Retrieving the correct French string from the hash array

The label controls in this report are references to library components. The Start() method for each label control component in the library retrieves the correct French string from the hash array:

```vbscript
Sub Start()
    Super::Start()
    ' Insert your code here
    Dim newText as String
    Dim labelName as String
    labelName=GetClassName(me)
    newText=labelDictionary.GetValue(labelName)
    If Not IsNull(newText) Then
        Text = newText
    End If
End Sub
```
The Start( ) method:

- Assigns the label control’s fully qualified component name to the local variable labelName.
- Calls labelDictionary’s GetValue method to retrieve the French string associated with the label control component and assigns this string to the local variable newText.
- Sets the label control component’s Text property to newText.
This chapter contains the following topics:

■ About report encoding
■ Running reports with e.Report Designer Professional
■ Working with encoding and Actuate Basic functions
■ Using the Actuate Basic Open statement
■ About Actuate Basic functions that require conversion to code page
■ Working with Actuate Basic source (.bas) file encoding
■ About database encoding
■ Designing Unicode reports
■ Controlling line breaking
About report encoding

Actuate uses UCS-2 encoding for internal processing. When an Actuate Basic report application passes strings to an external program that does not support UCS-2 or to the operating system, however, the strings are converted to code page. The code page is determined by, from highest precedence to lowest precedence:

- The top-level report component’s ReportEncoding property.
- The iServer System’s default encoding.
  For more information about setting the default iServer System encoding parameter, see Configuring BIRT iServer.
- The iServer System’s default locale.
  For more information about setting the default iServer System locale parameter, see Configuring BIRT iServer.
- The default locale defined in localemap.xml.
- ASCII
  If none of the above is set, the report uses ASCII encoding.

This code page is also used by Actuate Basic functions that operate on code page character codes and functions that operate on byte length.

The default code pages for supported locales are listed in Table 3-1.

Table 3-1  Default code pages for supported locales

<table>
<thead>
<tr>
<th>Locale</th>
<th>Code page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albanian</td>
<td>windows-1250</td>
</tr>
<tr>
<td>Arabic (Algeria)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Bahrain)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Egypt)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Iraq)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Jordan)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Kuwait)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Lebanon)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Libya)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Morocco)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Oman)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Qatar)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Locale</td>
<td>Code page</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Arabic (Saudi Arabia)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Syria)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Tunisia)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (U.A.E.)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Arabic (Yemen)</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>windows-1251</td>
</tr>
<tr>
<td>Chinese (Hong Kong SAR)</td>
<td>windows-950 (includes Hong Kong</td>
</tr>
<tr>
<td></td>
<td>Supplementary Character Set)</td>
</tr>
<tr>
<td>Chinese (PRC)</td>
<td>windows-936</td>
</tr>
<tr>
<td>Chinese (Singapore)</td>
<td>windows-936</td>
</tr>
<tr>
<td>Chinese (Taiwan)</td>
<td>windows-950 (includes Hong Kong</td>
</tr>
<tr>
<td></td>
<td>Supplementary Character Set)</td>
</tr>
<tr>
<td>Croatian</td>
<td>windows-1250</td>
</tr>
<tr>
<td>Czech</td>
<td>windows-1250</td>
</tr>
<tr>
<td>Danish (Denmark)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Dutch (Belgium)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Dutch (Netherlands)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>English (Australia)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>English (Belize)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>English (Canada)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>English (Ireland)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>English (New Zealand)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>English (South Africa)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>English (United Kingdom)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>English (United States)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Estonian</td>
<td>windows-1257</td>
</tr>
<tr>
<td>Farsi</td>
<td>windows-1256</td>
</tr>
<tr>
<td>Finnish</td>
<td>windows-1252</td>
</tr>
<tr>
<td>French (Canada)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>French (France)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>French (Switzerland)</td>
<td>windows-1252</td>
</tr>
</tbody>
</table>

(continues)
<table>
<thead>
<tr>
<th>Locale</th>
<th>Code page</th>
</tr>
</thead>
<tbody>
<tr>
<td>German (Austria)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>German (Germany)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>German (Liechtenstein)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>German (Switzerland)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Greek</td>
<td>windows-1253</td>
</tr>
<tr>
<td>Hebrew</td>
<td>windows-1255</td>
</tr>
<tr>
<td>Hungarian</td>
<td>windows-1250</td>
</tr>
<tr>
<td>Indonesian</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Italian (Italy)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Italian (Switzerland)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Japanese</td>
<td>windows-932</td>
</tr>
<tr>
<td>Korean</td>
<td>windows-949</td>
</tr>
<tr>
<td>Latvian</td>
<td>windows-1257</td>
</tr>
<tr>
<td>Norwegian (Bokmal)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Norwegian (Nynorsk)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Polish</td>
<td>windows-1250</td>
</tr>
<tr>
<td>Portuguese (Brazil)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Portuguese (Portugal)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Romanian</td>
<td>windows-1250</td>
</tr>
<tr>
<td>Russian</td>
<td>windows-1251</td>
</tr>
<tr>
<td>Serbian (Latin) (Yugoslavia)</td>
<td>windows-1251</td>
</tr>
<tr>
<td>Slovak</td>
<td>windows-1250</td>
</tr>
<tr>
<td>Slovenian</td>
<td>windows-1250</td>
</tr>
<tr>
<td>Spanish (Mexico)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Spanish (Spain)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Swedish (Finland)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Swedish (Sweden)</td>
<td>windows-1252</td>
</tr>
<tr>
<td>Thai</td>
<td>windows-874</td>
</tr>
<tr>
<td>Turkish (Turkey)</td>
<td>windows-1254</td>
</tr>
<tr>
<td>Ukrainian (Ukraine)</td>
<td>windows-1251</td>
</tr>
</tbody>
</table>
Actuate also supports the encodings listed in Table 3-2. For UTF-8, however, only the characters present in the UCS-2 character set are supported.

<table>
<thead>
<tr>
<th>Language family</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO Latin 1</td>
<td>ISO-8859-1</td>
</tr>
<tr>
<td>ISO Latin 2</td>
<td>ISO-8859-2</td>
</tr>
<tr>
<td>ISO Latin 3</td>
<td>ISO-8859-3</td>
</tr>
<tr>
<td>ISO Latin 4</td>
<td>ISO-8859-4</td>
</tr>
<tr>
<td>ISO Cyrillic</td>
<td>ISO-8859-5</td>
</tr>
<tr>
<td>ISO Arabic</td>
<td>ISO-8859-6</td>
</tr>
<tr>
<td>ISO Greek</td>
<td>ISO-8859-7</td>
</tr>
<tr>
<td>ISO Hebrew</td>
<td>ISO-8859-8</td>
</tr>
<tr>
<td>ISO Latin 5</td>
<td>ISO-8859-9</td>
</tr>
<tr>
<td>Simplified Chinese</td>
<td>EUC-CN</td>
</tr>
<tr>
<td>Traditional Chinese</td>
<td>EUC-TW (includes Hong Kong Supplementary Character Set)</td>
</tr>
<tr>
<td>Japanese</td>
<td>EUC-JP</td>
</tr>
<tr>
<td>Korean</td>
<td>EUC-KR</td>
</tr>
<tr>
<td>English</td>
<td>ASCII</td>
</tr>
<tr>
<td>Multiple code page</td>
<td>UTF-8</td>
</tr>
<tr>
<td>Multiple code page</td>
<td>UCS-2</td>
</tr>
</tbody>
</table>

Setting the ReportEncoding property

To set the ReportEncoding property, choose an encoding from the drop-down list in the top-level report component’s Component Editor, as shown in Figure 3-1.

Understanding the Language variable

Do not use the Language variable to specify a report’s encoding. The Language variable is provided for backward compatibility only.
Running reports with e.Report Designer Professional

When you run a report using e.Report Designer Professional, the encoding precedence is different from the encoding precedence on the iServer System.

On the desktop, a report’s encoding is determined by, from highest precedence to lowest precedence:

- The top-level report component’s ReportEncoding property.
- The default locale specified on the General page of the Options dialog.
- The default locale specified when the product is installed.
- The default locale defined in localemap.xml.
- ASCII
  If none of the above is set, the report uses ASCII encoding.

Working with encoding and Actuate Basic functions

Several Actuate Basic functions operate on character codes and byte length:

- Working with functions that operate on UCS-2 character codes
- Working with functions that operate on code page character codes
- Working with functions that operate on byte length
For more information about Actuate Basic functions, see *Programming with Actuate Basic*.

**Working with functions that operate on UCS-2 character codes**

Actuate uses UCS-2 encoding for internal processing. Several Actuate Basic functions operate on UCS-2 character codes.

**Using the AscW function**

The AscW function takes a string expression and returns the UCS-2 character code for the first character. For example:

```
AscW("A") returns 65
```

**Using ChrW and ChrW$ functions**

TheChrW and ChrW$ functions take a UCS-2 character code and return the character. For example:

```
ChrW$(65) returns A
ChrW(947) returns γ
```

**Using StringW and StringW$ functions**

The StringW and StringW$ functions take a numeric expression and a UCS-2 character code and return a string that contains the character repeated the specified number of times. For example:

```
StringW$(10,"#") returns #######
StringW$(10,947) returns γγγγγγγγγγ
```

**Working with functions that operate on code page character codes**

Several Actuate Basic functions operate on character codes in the runtime code page. The runtime code page is determined as described in “About report encoding,” earlier in this chapter. These functions are provided for backward compatibility only. If the report’s encoding is UCS-2, use the corresponding W functions.

**Using the Asc function**

The Asc function takes a string expression and returns the character code in the runtime code page for the first character in the string. The first character in the string must be present in the runtime code page. For example, if the first character
in the string is a Japanese character and the runtime code page is ASCII, the Asc function does not yield a meaningful result. For example:

```
Asc("ABC") returns 65
```

**Using Chr and Chr$ functions**

The Chr and Chr$ functions take a character code in the runtime code page and return the character. For example:

```
Chr$(65) returns A
Chr(227) returns â if runtime code page is Western European
Chr(227) returns γ if runtime code page is Greek
```

**Using the String$ function**

The String$ function takes a numeric expression and a character code in the runtime code page and returns a string that contains the character repeated the specified number of times. For example:

```
String$(5,65) returns AAAAA
String$(5,227) returns γyyyy if runtime code page is Greek
String$(5,227) returns âââââ if runtime code page is Western European
```

**Working with functions that operate on byte length**

Because Actuate uses UCS-2 encoding for internal processing, using functions that operate on byte length is not recommended. These functions are provided for backward compatibility only, and are described in Table 3-3. The byte length is determined by converting from UCS-2 to the runtime code page. The runtime code page is determined as described in “About report encoding,” earlier in this chapter. Passing a string that contains characters from multiple code pages does not yield a meaningful result.

If the runtime encoding is UCS-2:

- Every character is two bytes. For example:
  
  `LenB("ABC") returns 6`

- Actuate Basic aligns the character position to the character boundary. For example:
  
  `MidB("Widget",4,5) returns "idg"

  The starting position, 4, and the length, 5, point to the middle of a character, so Actuate Basic instead executes:
  
  `MidB("Widget",3,6)`
The starting position is 3, meaning the second character. The function returns 6 bytes, or three characters.

**Table 3-3** Functions that operate on byte length

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputB</td>
<td>Returns a specified number of bytes from a sequential file.</td>
</tr>
<tr>
<td>InstrB</td>
<td>Returns the starting byte of the occurrence of one string within another.</td>
</tr>
<tr>
<td>LeftB, LeftB$</td>
<td>Returns a segment of a Variant or String, starting at the byte that is furthest to the left.</td>
</tr>
<tr>
<td>LenB</td>
<td>Returns the number of bytes in a string expression.</td>
</tr>
<tr>
<td>MidB, MidB$</td>
<td>Returns specified portion of a string expression.</td>
</tr>
<tr>
<td>RightB, RightB$</td>
<td>Returns a segment of a Variant or String, starting at the byte that is furthest to the right and working toward the left.</td>
</tr>
</tbody>
</table>

**Using the Actuate Basic Open statement**

Use the Actuate Basic Open statement to open a text file. Use the Open statement’s Encoding parameter to specify the file’s encoding. Both UCS-2LE (Little Endian) and UCS-2BE (Big Endian) are supported. If the file contains only English characters, set the Encoding parameter to ASCII.

If the encoding name is text (case-insensitive), the operating system code page is used. If the file’s encoding is not specified or the encoding name is not valid, the report’s runtime encoding is used.

The file name must use the operating system code page.

For example:

```vbscript
Function Start( ) As Boolean
    Start = Super::Start( )

    Open "Korean.txt" For Input "windows-949" As #1
    Open "KoreanOut.txt" For Output "windows-949" As #2
    Open "Japanese.txt" For Input "windows-932" As #3
    Open "JapaneseOut.txt" For Output "windows-932" As #4
    Open "Chinese.txt" For Input "windows-936" As #5
    Open "ChineseOut.txt" For Output "windows-936" As #6

(continues)
```
Open "Unicode.txt" For Input "UCS-2LE" As #7
Open "UnicodeOut.txt" For Output "UCS-2LE" As #8
End Function

For more information about the Open statement, see Programming with Actuate Basic.

About Actuate Basic functions that require conversion to code page

The following Actuate Basic functions require that strings be converted from UCS-2, Actuate’s internal encoding, to a code page. The code page is determined as described in “About report encoding,” earlier in this chapter.

■ About Environ and Environ$ functions
■ About the Shell function
■ About functions that call external C or C++ functions
■ About functions that access operating system resources

For more information about Actuate Basic functions, see Programming with Actuate Basic.

About Environ and Environ$ functions

The Environ[$] function takes the name of an environment variable and returns its setting. The name of the environment variable is converted from UCS-2 to code page. The returned value is converted from code page to UCS-2.

About the Shell function

The Shell function runs a program. The name of the executable and any parameters are converted from UCS-2 to code page. Every character in the executable and parameter names must be present in the code page.

About functions that call external C or C++ functions

Using Actuate Basic, a report developer can call external C or C++ functions stored in a DLL or a shared library. Strings are passed to external functions as char * data type. Strings are converted from UCS-2 to the encoding returned by the interface char * AcGetDllEncoding( ). If this interface does not exist or returns an invalid value, the encoding is determined as described in “About report encoding,” earlier in this chapter. If this interface returns UCS-2, strings are not converted.
If necessary, returned strings are converted back to UCS-2.

**About functions that access operating system resources**

Several Actuate Basic functions access operating system resources, for example:

- ChDir
- Open
- Kill

The parameter names for these functions are converted from UCS-2 to code page. Every character in the parameter names must be present in the code page.

---

**Working with Actuate Basic source (.bas) file encoding**

If your Actuate Basic report designs contain characters from multiple code pages, you must save the Actuate Basic source (.bas) files with UCS-2LE encoding.

**Understanding Actuate Basic language elements**

The following Actuate Basic language elements cannot contain characters from multiple code pages:

- Keywords and built-in operators
- Built-in data types
- Identifiers defined in the Actuate Foundation Class library

The following Actuate Basic language elements can contain characters from multiple code pages:

- User-defined identifiers such as:
  - Class names
  - Method and function names
  - Variable names
- String literals
- Comments in Actuate Basic code
Saving Actuate Basic source (.bas) files

The report developer can save Actuate Basic source (.bas) files as:

- Text (operating system code page)
  If you save an Actuate Basic source file as Text, every character in the source file must be present in the operating system code page.

- Unicode (UCS-2LE)

Actuate Basic source files include

- Internal Actuate Basic source files
  An internal Actuate Basic source file is generated from an ROD and is used to create an ROX.

- External Actuate Basic source files
  External Actuate Basic source files are created by the report developer as part of the report design.

If you are connecting to a database with table and column names that contain characters from multiple code pages, you must save the internal Actuate Basic source file with UCS-2LE encoding. For information about connecting to a Unicode database, see Configuring BIRT iServer.

If your report design includes Actuate Basic source files that contain characters from multiple code pages, you must save the external Actuate Basic source files with UCS-2LE encoding.

How to specify the default encoding for Actuate Basic source files

1. In e.Report Designer Professional, choose Tools ➤ Options.
2. In Options, choose General.
3. In Internal Basic source encoding or External Basic source encoding, choose Text or Unicode (UCS-2LE), as shown in Figure 3-2. Choose OK.

About Windows platform limitations

If your reports contain characters from multiple code pages, the reports must be developed, generated, and viewed on a supported Windows platform.

About database encoding

You must configure your database clients to support the encoding used by the database. For more information about configuring database clients, see Configuring BIRT iServer.
Setting NLS_LANG for an Oracle database

If an Actuate server running on a UNIX platform connects to an Oracle database, the system administrator must set NLS_LANG in pmd11.sh, for example:

```
export NLS_LANG
NLS_LANG="AMERICAN_AMERICA.UTF8"
```

The Actuate server’s default encoding must match the setting of NLS_LANG. For example, if NLS_LANG is set to AMERICAN_AMERICA.UTF8, the Actuate server’s default encoding must be set to UTF-8. For more information about setting the default iServer System encoding parameter, see Configuring BIRT iServer.

Setting LC_ALL for a Sybase database

When an Actuate server connects to a Sybase database running on a UNIX platform, the Actuate server uses the value of LC_ALL to determine the encoding
of database strings. Table 3-4 lists the supported values of LC_ALL for Sybase on AIX, HP-UX, and Solaris platforms.

<table>
<thead>
<tr>
<th>AIX</th>
<th>HP-UX</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm-1370</td>
<td>ibm-1051</td>
<td>ibm-1383</td>
</tr>
<tr>
<td>ibm-1383</td>
<td>ibm-1383</td>
<td>ibm-1386</td>
</tr>
<tr>
<td>ibm-1386</td>
<td>ibm-1386</td>
<td>ibm-33722</td>
</tr>
<tr>
<td>ibm-33722</td>
<td>ibm-33722</td>
<td>ibm-915</td>
</tr>
<tr>
<td>ibm-943_P14A-2000</td>
<td>ibm-970</td>
<td>ibm-970</td>
</tr>
<tr>
<td>ibm-970</td>
<td>LATIN_1</td>
<td>LATIN_1</td>
</tr>
<tr>
<td>LATIN_1</td>
<td>UTF8</td>
<td>UTF8</td>
</tr>
<tr>
<td>UTF8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sybase Open Client 12.5 does not support UNICHAR and UNIVARCHAR fields from a Sybase database.

**Designing Unicode reports**

If your Actuate Basic reports and report designs contain characters from multiple code pages, you must:

- Develop your reports on a supported Windows platform.
- Install a Unicode font such as Arial Unicode MS on your system.
- Configure your database clients to support Unicode data. For more information about connecting to a Unicode database, see *Configuring BIRT iServer.*
- Specify a Unicode font for labels and data.
- Specify a Unicode font for code.
- Save the Actuate Basic source (.bas) files with UCS-2LE encoding.
Controlling line breaking

You can control the line breaking for text in Actuate Basic reports by specifying characters that should not appear at the beginning or the end of a line. To control line breaking, add the following Actuate iServer configuration variables:

- **TurnOnAsianLineBreakingRule**
  Set to 1 to enable line breaking or 0 to disable line breaking.

- **DoNotBegin**
  List characters that should not appear at the beginning of a line. By default, the following characters do not appear at the beginning of a line if line breaking is enabled:

  , . ? ! ) ] } > \x3001 \x3002 \xff0e \xff64 \xff61 \xff1f \xff01 \xff09 \xff3d \xff5d \xff1e \x309c \xff9f \x309b \xff9e \x300d \xff63 \x3015 \x300b \x300f \x3011 \x30fc \xff70 \xff1b

- **DoNotEnd**
  List characters that should not appear at the end of a line. By default, the following characters do not appear at the end of a line if line breaking is enabled:

  ( [ { < \xff08 \xff3b \xff5b \xff1c \x300c \xff62 \x3014 \x300a \x300e \x3010

For example:

```plaintext
TurnOnAsianLineBreakingRule = 1
DoNotBegin = ", . ? ! ) ] } > \x3001 \x3002 \xff0e \xff64 \xff61 \xff1f \xff01 \xff09 \xff3d \xff5d \xff1e \x309c \xff9f \x309b \xff9e \x300d \xff63 \x3015 \x300b \x300f \x3011 \x30fc \xff70 \xff1b"
DoNotEnd = "( [ { < \xff08 \xff3b \xff5b \xff1c \x300c \xff62 \x3014 \x300a \x300e \x3010"
```

The entries preceded by a backslash (\) are hexadecimal values.

If your Actuate Basic reports contain dynamic text controls, you must also do the following:

1. Add these variables to the registry under HK_CURRENT_USER/Software/Actuate/e.Report Designer Professional 11.0/Settings.
2. Recompile the reports.
Chapter 4

Using fonts in reports with multiple locales

This chapter contains the following topics:

- Using externalized fonts
- Using PostScript Type1 fonts
- Font embedding in PDF output
- Using fonts in controls
- Installing printer fonts on UNIX platforms
- Printing dynamic text controls on a UNIX printer
- Using Unicode fonts
Using externalized fonts

Actuate uses font metrics to determine font characteristics, such as character widths and heights, which it uses to compute line breaks, text truncation, fill characters, and so on. Actuate requires font metrics in the following situations:

- Rendering DHTML reports
- Rendering PDF reports
- Generating dynamic text control data
- Generating Excel data as specified by AFC Excel API code used in the report design

When the iServer System renders or generates Actuate Basic report output, it uses the font metrics in the master fonts file or in the report executable (ROX). The Actuate server looks for font metrics in the ROX if they are not present in the master fonts file. The master fonts file is an Actuate Basic report executable located in \Program Files\Actuate11\iServer\etc\master_fonts.rox on Windows XP platforms and $AC_SERVER_HOME/etc/master_fonts.rox on UNIX platforms.

You can instruct the iServer System to look for font metrics in the ROX before looking in the master fonts file, or you can instruct the iServer System not to look for font metrics in the master fonts file. To make either of these changes, you must modify the file acserverconfig.xml.

How to modify acserverconfig.xml

1. Using Configuration Console, stop the Actuate server or cluster.

2. Open \Program Files\Actuate11\iServer\etc\acserverconfig.xml or $AC_SERVER_HOME/etc/acserverconfig.xml in a text editor.

3. Modify acserverconfig.xml.

   Add the UseExternalizedFonts variable to the list of System variables. By default, the UseExternalizedFonts variable is set to Primary. Primary means that the Actuate server looks for font metrics in the master fonts file before looking in the ROX.

   If you want the Actuate server to look for font metrics in the ROX before looking in the master fonts file, set UseExternalizedFonts to Secondary. If you do not want the Actuate server to look for font metrics in the master fonts file, set UseExternalizedFonts to No.
For example:

```
<Config>
  <System
    LicenseKey="xxxxx-xxxxx-xxxxx-xxxx"
    SystemName="MySystem"
    DefaultLocale="en_US"
    DefaultEncoding="windows-1252"
    SystemDefaultVolume="MyVolume"
    ...
    UseExternalizedFonts="No">
    ...
    ...
  </System>
  ...
</Config>
```

4 Save acserverconfig.xml.

5 Using Configuration Console, restart the Actuate server or cluster.

A report developer can create a customized master fonts file that contains fonts that are not included in master_fonts.rox. If the same font is included in both the customized master fonts file and master_fonts.rox, the font metrics in the customized master fonts file take precedence. The customized master fonts file must be called customized_fonts.rox and must be placed in `\Program Files\Actuate11\iServer\etc` or `$AC_SERVER_HOME/etc` for every Actuate server in the cluster. If you are using a customized master fonts file that contains all the fonts you use, performance may improve slightly if you rename master_fonts.rox. For more information about creating a customized master fonts file, see Developing Reports using e.Report Designer Professional.

**Using PostScript Type1 fonts**

e.Report Designer Professional does not embed font width information for PostScript Type1 fonts in the ROX. If you use Type1 fonts in your report designs, you may observe the following:

- In PDF output, right-aligned and centered controls are not displayed correctly.
- In DHTML output, right-aligned and centered dynamic text controls are not displayed correctly.
For this reason, it is recommended that you use TrueType ttf or ttc fonts in your report designs.

Actuate does not support PostScript Type2, Type3, or Type4 fonts.

---

**Font embedding in PDF output**

This topic describes font embedding with the PDF Converter. If you are using a render profile to specify PDF output, see *Developing Reports using e.Report Designer Professional*.

By default, the Actuate iServer View process does not embed many fonts in an Actuate Basic report’s PDF output. You can, however, override the default behavior.

**Default font embedding in PDF output**

For Latin 1 and CJK languages, fonts are not embedded in PDF output. Actuate supports the following Latin 1 languages:

- Albanian
- Danish
- Dutch
- English
- Finnish
- French
- German
- Icelandic
- Indonesian
- Italian
- Norwegian
- Portuguese
- Spanish
- Swedish

Actuate supports the following CJK languages:

- Chinese
- Japanese
Korean
For languages other than Latin 1 and CJK languages, fonts are embedded in PDF output.

**Overriding default font embedding in PDF output**

You can ensure that an Actuate Basic report is displayed in Acrobat Reader using the fonts with which it was designed. To do so, embed the fonts or font subsets in the report’s PDF output. You can embed any UCS-2 character as long as the font is of one of the following types:

- TrueType with MS Unicode encoding
- TrueType collection
- OpenType

Before you can embed a font or a subset of a font, you must map the font.

**Mapping a font**

To map a font, you enter the font face and style and the name of the corresponding font file in a file called pdffont.map. You must map a font if either of the following statements is true:

- You want to embed the font or a subset of the font in an Actuate Basic report’s PDF output.
- The Actuate iServer is installed on a UNIX platform and your Actuate Basic reports use TrueType fonts.

**How to map a font**

1. Stop the Actuate server.
2. If the Actuate server is installed on a UNIX platform, create a directory in $AC_SERVER_HOME, for example $AC_SERVER_HOME/ttfont, and place the font files in this directory.
3. Specify the directory that contains the font files:
   1. In Configuration Console, choose Servers ➤ Advanced.
   3. Type the name of the font directory. Choose OK.
4. Open the text file pdffont.map in $AC_SERVER_HOME/etc.
pdffont.map maps the font face and style used in an Actuate Basic report design to the font file name. By default, pdffont.map contains the following entries:

Arial ARIAL.TTF
ArialBold ARIALBD.TTF
ArialBoldItalic ARIALBI.TTF
ArialItalic ARIALI.TTF
PalatinoLinotype pala.TTF
PalatinoLinotypeBold palab.TTF
PalatinoLinotypeBoldItalic palabi.TTF
PalatinoLinotypeItalic palai.TTF
MSGothic MSGOTHIC.TTC
MSPGothic MSGOTHIC.TTC
MSUIGothic MSGOTHIC.TTC
ArialUnicodeMS ARIALUNI.TTF

Add the required mappings to pdffont.map.

The following rules apply:

- Your text editor must support UTF-8 encoding.
- The name of the font face and style is the first item on the line. The font face name must exactly match the font face in the Actuate Basic report design, except that it must not contain spaces. This name is case-sensitive. For example, if the report design uses SimSun, the entry in pdffont.map must be SimSun, not Simsun or simsun.
- You can append Bold, Italic, or BoldItalic to the font face. If the font face contains the word strong, replace it with Bold. If the font face contains the word oblique, replace it with Italic.
- The font’s file name is the second item on the line. The font file name must exactly match the font file name in the font directory.
- Each font in a TrueType collection (TTC) must be listed as a separate entry.

Restart the Actuate server.

Embedding a font

You can embed a font in an Actuate Basic report’s PDF output. The font file must contain the UCS-2 character map. All styles (Regular, Bold, Italic, BoldItalic) of the font are embedded, provided you map a separate TTF font file for each style. For each embedded font, the size of the PDF output increases by approximately 200 KB.

Symbol fonts cannot be embedded.

Before you can embed a font, you must map the font.
How to embed a font in PDF output

1. If necessary, install the font on the Actuate server.

2. Using a text editor that supports UTF-8 encoding, add a line to $AC_SERVER_HOME/etc/cjk.conf.
   For example, to embed Palatino Linotype fonts, add the following line to the embed list in cjk.conf:
   
   PALATINOLINOTYPE : embed
   
   The name of the font face must be all upper case and must not contain spaces.

Embedding a subset of a font

In many cases, embedding an entire font causes an Actuate Basic report’s PDF output to be too large. To reduce the size of the PDF output, you can embed only the font information for the characters that appear in the report. Embedding a subset of a font increases PDF generation time.

Before you can embed a subset of a font, you must map the font. For Latin 1 and CJK languages, you must also embed the font.

How to embed a subset of a font in PDF output

1. If necessary, install the font on the Actuate server.

2. Using a text editor that supports UTF-8 encoding, add a line to $AC_SERVER_HOME/etc/cjk.conf.
   For example, to embed Palatino Linotype font subsets, add the following line to the subset list in cjk.conf:
   
   PALATINOLINOTYPE : subset
   
   The name of the font face must be all upper case and must not contain spaces.

Using fonts in controls

A report developer can specify fonts for controls. The report developer must use the English font name rather than the native language font name. For example, use Gulim rather than the Korean font name.
If the report developer does not specify the fonts for controls, the default fonts are used. The default fonts are determined by the locale used to generate the report. Table 4-1 lists the default fonts for different locales.

<table>
<thead>
<tr>
<th>Locale</th>
<th>Default fonts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese (Simplified)</td>
<td>simsun.ttc</td>
</tr>
<tr>
<td>Chinese (Traditional)</td>
<td>mingliu.ttc</td>
</tr>
<tr>
<td>Japanese</td>
<td>msgothic.ttc, msmincho.ttc</td>
</tr>
<tr>
<td>Korean</td>
<td>gulim.ttc, batang.ttc</td>
</tr>
<tr>
<td>Other</td>
<td>arial.ttf, arialbd.ttf, arialbi.ttf, ariali.ttf, times.ttf, timesbd.ttf, timesbi.ttf, timesi.ttf</td>
</tr>
</tbody>
</table>

**Windows platforms**

Fonts used in controls must be installed in the operating system’s Fonts folder.

**How to install a font on a Windows platform**

1. From the task bar, choose Start ➔ Settings ➔ Control Panel.
2. In Control Panel, double-click Fonts.
3. In Fonts, choose File ➔ Install New Font.
4. In Add Fonts, navigate to the folder that contains the font.
5. In List of fonts, select the font to install.
6. Check Copy fonts to Fonts folder and choose OK.

**UNIX platforms**

Fonts used in charts must be placed in the directory $JAVA_HOME/jre/lib/fonts. For example, if you are using the JRE installed with the Actuate server, place the fonts in $AC_SERVER_HOME/jre/lib/fonts.
Installing printer fonts on UNIX platforms

Before you install a font on an Actuate server running on a UNIX platform, you must use a third-party tool such as Fontographer to:

- Convert the TTF or TTC font to a PostScript font
- Generate the font’s AFM and PFA files
  
PFA files are not generated for Chinese, Japanese, and Korean fonts.

You can then use the Actuate utility fontutils to install the PostScript font and make it available to the Actuate server for printing reports. fontutils is located in $AC_SERVER_HOME/bin. Make sure the path

$AC_SERVER_HOME/lib

is appended to the library path environment variable:

- On SunOS the environment variable is LD_LIBRARY_PATH
- On AIX the environment variable is LIBPATH
- On HP-UX the environment variable is SHLIB_PATH

For more information about fontutils, see Configuring BIRT iServer.

**How to install a Chinese, Japanese, or Korean font on an Actuate server**

Because they are not embedded in the PostScript file, Chinese, Japanese, and Korean fonts must also be installed on the printer.

1. Copy the font’s .afm file to $AC_SERVER_HOME/bin.
2. If necessary, convert the .afm file from DOS to ISO format using a utility such as dos2unix.
3. Change the file’s permissions so that it is readable by all users.
4. On the command line type

   fontutils -T 1 -l <Windows_font_name> -2 <font_file>.afm -3 dummy.pfa -4 <code_page>

   where

   - <Windows_font_name> is the name of the font used in the report design, including bold and italic information, for example:
     - Gulim
     - Gulim-Bold
     - Gulim-Italic
     - Gulim-BoldItalic
Working in Multiple Locales using Actuate Basic Technology

<Windows_font_name> must be an ASCII string.

- <font_file> is the name of the .afm file.
- dummy.pfa is a dummy file name.
- <code_page> is the font’s code page. The following code pages are supported:
  - Code page 932 (Japanese)
  - Code page 936 (Simplified Chinese)
  - Code page 949 (Korean)
  - Code page 950 (Traditional Chinese)

For example, to install the Gulim-Bold font using the file Gulim.afm:

```
fontutils -T 1 -1 Gulim-Bold -2 Gulim.afm -3 dummy.pfa -4 949
```

**How to install other fonts on an Actuate server**

Because they are embedded in the PostScript file, these fonts do not have to be installed on the printer.

1. Copy the font’s .afm and .pfa files to $AC_SERVER_HOME/bin.
2. If necessary, convert the .afm file and the .pfa file from DOS to ISO format using a utility such as dos2unix.
3. Change the files’ permissions so that they are readable by all users.
4. Change the .pfa file’s permissions so that it is writable by all users.
5. On the command line type

```
fontutils -T 1 -1 <Windows_font_name> -2 <font_file>.afm -3 <font_file>.pfa -4 <code_page>
```

where

- <Windows_font_name> is the name of the font used in the report design, including bold and italic information, for example:
  - Century
  - Century-Bold
  - Century-Italic
  - Century-BoldItalic
- <code_page> must be an ASCII string.
- <font_file> is the name of the .afm or .pfa file.
- <code_page> is the font’s code page. The following code pages are supported:
- Code page 874 (Thai)
- Code page 1250 (Latin 2)
- Code page 1251 (Cyrillic)
- Code page 1252 (Latin 1)
- Code page 1253 (Greek)
- Code page 1254 (Latin 5)
- Code page 1255 (Hebrew)
- Code page 1256 (Arabic)
- Code page 1257 (Baltic)

For example, to install the Century-Bold font using the files Century.afm and Century.pfa:

```
fontutils -T 1 -1 Century-Bold -2 Century.afm -3 Century.pfa -4 1252
```

How to uninstall a PostScript font

On the command line type

```
fontutils -T 2 -1 <Windows_font_name> -4 <code_page>
```

where

- `<Windows_font_name>` is the name of the font used in the report design, including bold and italic information, for example:
  - Century
  - Century-Bold
  - Century-Italic
  - Century-BoldItalic
- `<Windows_font_name>` must be an ASCII string.
- `<code_page>` is the font’s code page.

For example, to uninstall the Century-Bold font:

```
fontutils -T 2 -1 Century-Bold -4 1252
```

Printing dynamic text controls on a UNIX printer

If a report design contains a dynamic text control and the report will be printed on a UNIX system printer, the report developer should allow for text expansion. For example, if the report developer uses the font MS Mincho, the font Ryumin is
substituted when the report is printed on a UNIX system printer. In the font Ryumin, Arabic numbers are wider than they are in MS Mincho. If the dynamic text control contains Arabic numbers, it may not be wide enough to accommodate the numbers when the report is printed. To correct the problem, increase the value of the Text Layout ➔ LineWidthPadding property.

Using Unicode fonts

In US English e.Report Designer Professional, the default fonts are as follows:

- For labels, Arial 12-point Black Western
- For data, Times New Roman 12-point Black Western
- For code, Courier New 10-point Black Western

If your report designs use label controls or data controls that have instances that contain characters from multiple code pages, you must change the default Label Font or Data Font to a Unicode font such as Arial Unicode MS or Lucida Sans Unicode. If, however, individual controls have instances that contain characters from a single code page, it is not necessary to change the default font to a Unicode font; you can set the font for each control individually. For example, a report design has three controls: ChineseControl, JapaneseControl, and KoreanControl. Each control has instances that contain characters from a single code page. For each control, set the Font properties accordingly.

If your Actuate Basic code contains characters from multiple code pages, you must change the default Source Editor Font to a Unicode font such as Arial Unicode MS or Lucida Sans Unicode.

How to change the default font

1. If you are changing the default font for code, close any open Actuate Basic source (.bas) files.
3. In Options, choose Fonts.
4. In Label font, Data font, or Source editor font, choose Change.
5. In Font, set the following formats, and choose OK:
   - Font
   - Font style
   - Size
   - Effects
   - Color
Script

Figure 4-1 shows an example of how to specify font formatting.

Figure 4-1  Specifying font formatting

6  In Options, choose OK.
This chapter contains the following topics:

- About right-to-left orientation
- Displaying the application window with right-to-left orientation
- About the right-to-left Design Editor window
- Displaying reports with right-to-left orientation
- Changing the contents of controls for right-to-left reports
About right-to-left orientation

You can design Actuate Basic reports with right-to-left orientation in e.Report Designer Professional. You can also display the e.Report Designer Professional and Design Editor windows with right-to-left orientation.

Displaying the application window with right-to-left orientation

By default, the e.Report Designer Professional window displays with right-to-left orientation on right-to-left operating systems such as Arabic, Farsi, and Hebrew localized Windows 98 and localized Windows 2000. You can change the orientation on other Windows 2000 operating systems to right-to-left by setting the WindowOrientation registry key or by using a command line option. Most dialog boxes, however, display with left-to-right orientation even if the e.Report Designer Professional window displays with right-to-left orientation.

How to display the application window with right-to-left orientation

Set the WindowOrientation registry key:

1. Exit e.Report Designer Professional.
2. On the task bar, choose Start ➤ Run.
   The Run dialog box appears.
   Type
   regedit.exe
3. Choose OK.
   The Registry Editor window appears.
4. Navigate to HKEY_CURRENT_USER\Software\Actuate\e.Report Designer Professional 11.0\Settings.
5. Choose Edit ➤ New ➤ String Value.
   Type
   WindowOrientation
6. Press Enter.
7. Choose Edit ➤ Modify.
   The Edit String dialog box appears.
   Type RTL in the Value data text box.
About the right-to-left Design Editor window

The orientation of the Design Editor window is the same as the orientation of the e.Report Designer Professional window. In a right-to-left Design Editor window, the structure pane is displayed on the right and the layout pane is displayed on the left.

Controls in the layout pane are positioned relative to the layout pane’s origin. For right-to-left orientation, the origin of the layout pane is the upper right corner. If the orientation is changed to left-to-right, the controls maintain their position relative to the origin, but the origin changes from the upper right corner to the upper left corner.

For example, a control that appears in the upper right corner with right-to-left orientation appears in the upper left corner with left-to-right orientation. The contents of the controls do not change.

Displaying reports with right-to-left orientation

By default, a report displays with left-to-right orientation even if the viewer displays with right-to-left orientation. To display a report with right-to-left
orientation, set the top-level report component’s LayoutOrientation property to RightToLeft, as shown in Figure 5-1.

Figure 5-1 Selecting right-to-left orientation

The setting of LayoutOrientation determines the orientation regardless of operating system or locale. Any subreports are displayed with the same orientation as the top-level report.

About positioning controls in a report

Controls in a report are positioned relative to the report’s origin. For left-to-right orientation, the origin of the report is the upper left corner. If the orientation is changed to right-to-left, the controls maintain their position relative to the origin, but the origin changes from the upper left corner to the upper right corner.

For example, a control that appears in the upper left corner with left-to-right orientation appears in the upper right corner with right-to-left orientation at view time. The contents of the controls do not change.

The following illustrations show a report with different orientations. Figure 5-2 shows the report with left-to-right orientation. Figure 5-3 shows the report with right-to-left orientation.

Figure 5-2 Example of a report with left-to-right orientation
Chapter 5, Designing reports with right-to-left orientation

Setting the orientation programmatically

You can set the right-to-left orientation programmatically with the `SetLayoutOrientation()` method defined on the `AcReport` class.

For example, assume you have a text control that contains the report's locale. You can override the control's `Finish` method to set the orientation to right-to-left for an Arabic report:

```vba
Sub Finish
    Super::Finish()
    ' Set report layout orientation to right-to-left for Arabic locale
    If DataValue = "Arabic" Then
        Container.GetReport().SetLayoutOrientation(RightToLeft)
    Else
        Container.GetReport().SetLayoutOrientation(LeftToRight)
    End If
End Sub
```

You can get the orientation with the `GetLayoutOrientation()` method defined on the `AcReport` class. Both `SetLayoutOrientation()` and `GetLayoutOrientation()` can be called at report generation time, but not at view time.

Changing the contents of controls for right-to-left reports

When the orientation of a report is changed to right-to-left, the controls in the report maintain their position relative to the origin, but the contents of the controls do not change. You can, however, change the contents of image, chart, and textual controls using other techniques:

- Image controls
To change the orientation of an image, modify the image file or change the image programmatically. You can change the orientation of bitmap images using Microsoft Developer Studio.

■ Chart controls
To change the orientation of a chart, use the overlay axis instead of the y-axis. Sort the labels for the x-axis and the values for the overlay axis in the database or in Actuate Basic so that the labels and values read from right to left. For information about the overlay axis, see Developing Reports using e.Report Designer Professional.

■ Textual controls
Change the setting of the TextPlacement-Horizontal property at report generation time using Actuate Basic. For example, you might change the setting for text controls from TextAlignLeft to TextAlignRight for right-to-left reports.
Use the locale codes in Table A-1 to identify locales in Actuate Basic code.

<table>
<thead>
<tr>
<th>Locale</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albanian</td>
<td>sq_AL</td>
</tr>
<tr>
<td>Arabic (Algeria)</td>
<td>ar_DZ</td>
</tr>
<tr>
<td>Arabic (Bahrain)</td>
<td>ar_BH</td>
</tr>
<tr>
<td>Arabic (Egypt)</td>
<td>ar_EG</td>
</tr>
<tr>
<td>Arabic (Iraq)</td>
<td>ar_IQ</td>
</tr>
<tr>
<td>Arabic (Jordan)</td>
<td>ar_JO</td>
</tr>
<tr>
<td>Arabic (Kuwait)</td>
<td>ar_KW</td>
</tr>
<tr>
<td>Arabic (Lebanon)</td>
<td>ar_LB</td>
</tr>
<tr>
<td>Arabic (Libya)</td>
<td>ar_LY</td>
</tr>
<tr>
<td>Arabic (Morocco)</td>
<td>ar_MA</td>
</tr>
<tr>
<td>Arabic (Oman)</td>
<td>ar_OM</td>
</tr>
<tr>
<td>Arabic (Qatar)</td>
<td>ar_QA</td>
</tr>
<tr>
<td>Arabic (Saudi Arabia)</td>
<td>ar_SA</td>
</tr>
<tr>
<td>Arabic (Syria)</td>
<td>ar_SY</td>
</tr>
<tr>
<td>Arabic (Tunisia)</td>
<td>ar_TN</td>
</tr>
<tr>
<td>Arabic (U.A.E.)</td>
<td>ar_AE</td>
</tr>
</tbody>
</table>

(continues)
<table>
<thead>
<tr>
<th>Locale</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic (Yemen)</td>
<td>ar_YE</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>bg_BG</td>
</tr>
<tr>
<td>Chinese (Hong Kong SAR)</td>
<td>zh_HK</td>
</tr>
<tr>
<td>Chinese (PRC)</td>
<td>zh_CN</td>
</tr>
<tr>
<td>Chinese (Singapore)</td>
<td>zh_SG</td>
</tr>
<tr>
<td>Chinese (Taiwan)</td>
<td>zh_TW</td>
</tr>
<tr>
<td>Croatian</td>
<td>hr_HR</td>
</tr>
<tr>
<td>Czech</td>
<td>cs_CZ</td>
</tr>
<tr>
<td>Danish (Denmark)</td>
<td>da_DK</td>
</tr>
<tr>
<td>Dutch (Belgium)</td>
<td>nl_BE</td>
</tr>
<tr>
<td>Dutch (Netherlands)</td>
<td>nl_NL</td>
</tr>
<tr>
<td>English (Australia)</td>
<td>en_AU</td>
</tr>
<tr>
<td>English (Belize)</td>
<td>en_BZ</td>
</tr>
<tr>
<td>English (Canada)</td>
<td>en_CA</td>
</tr>
<tr>
<td>English (Ireland)</td>
<td>en_IE</td>
</tr>
<tr>
<td>English (New Zealand)</td>
<td>en_NZ</td>
</tr>
<tr>
<td>English (South Africa)</td>
<td>en_ZA</td>
</tr>
<tr>
<td>English (United Kingdom)</td>
<td>en_GB</td>
</tr>
<tr>
<td>English (United States)</td>
<td>en_US</td>
</tr>
<tr>
<td>Estonian</td>
<td>et_EE</td>
</tr>
<tr>
<td>Farsi</td>
<td>fa_IR</td>
</tr>
<tr>
<td>Finnish</td>
<td>fi FI</td>
</tr>
<tr>
<td>French (Canada)</td>
<td>fr_CA</td>
</tr>
<tr>
<td>French (France)</td>
<td>fr_FR</td>
</tr>
<tr>
<td>French (Switzerland)</td>
<td>fr_CH</td>
</tr>
<tr>
<td>German (Austria)</td>
<td>de_AT</td>
</tr>
<tr>
<td>German (Germany)</td>
<td>de_DE</td>
</tr>
<tr>
<td>German (Liechtenstein)</td>
<td>de_LI</td>
</tr>
<tr>
<td>German (Switzerland)</td>
<td>de_CH</td>
</tr>
<tr>
<td>Greek</td>
<td>el_GR</td>
</tr>
<tr>
<td>Hebrew</td>
<td>he_IL</td>
</tr>
<tr>
<td>Locale</td>
<td>Code</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Hebrew</td>
<td>iw_IL (Use only with Actuate iServer Components for BEA WebLogic Workshop.)</td>
</tr>
<tr>
<td>Hungarian</td>
<td>hu_HU</td>
</tr>
<tr>
<td>Indonesian</td>
<td>id_ID</td>
</tr>
<tr>
<td>Indonesian</td>
<td>in_ID (Use only with Actuate iServer Components for BEA WebLogic Workshop.)</td>
</tr>
<tr>
<td>Italian (Italy)</td>
<td>it_IT</td>
</tr>
<tr>
<td>Italian (Switzerland)</td>
<td>it_CH</td>
</tr>
<tr>
<td>Japanese</td>
<td>ja_JP</td>
</tr>
<tr>
<td>Korean</td>
<td>ko_KR</td>
</tr>
<tr>
<td>Latvian</td>
<td>lv_LV</td>
</tr>
<tr>
<td>Norwegian (Bokmal)</td>
<td>no_NO</td>
</tr>
<tr>
<td>Norwegian (Nynorsk)</td>
<td>no_NY</td>
</tr>
<tr>
<td>Polish</td>
<td>pl_PL</td>
</tr>
<tr>
<td>Portuguese (Brazil)</td>
<td>pt_BR</td>
</tr>
<tr>
<td>Portuguese (Portugal)</td>
<td>pt_PT</td>
</tr>
<tr>
<td>Romanian</td>
<td>ro_RO</td>
</tr>
<tr>
<td>Russian</td>
<td>ru_RU</td>
</tr>
<tr>
<td>Serbian (Latin) (Yugoslavia)</td>
<td>sr_YU</td>
</tr>
<tr>
<td>Slovak</td>
<td>sk_SK</td>
</tr>
<tr>
<td>Slovenian</td>
<td>sl_SI</td>
</tr>
<tr>
<td>Spanish (Mexico)</td>
<td>es_MX</td>
</tr>
<tr>
<td>Spanish (Spain)</td>
<td>es_ES</td>
</tr>
<tr>
<td>Swedish (Finland)</td>
<td>sv_FI</td>
</tr>
<tr>
<td>Swedish (Sweden)</td>
<td>sv_SE</td>
</tr>
<tr>
<td>Thai</td>
<td>th_TH</td>
</tr>
<tr>
<td>Turkish (Turkey)</td>
<td>tr_TR</td>
</tr>
<tr>
<td>Ukrainian (Ukraine)</td>
<td>uk_UA</td>
</tr>
</tbody>
</table>
Working in Multiple Locales using Actuate Basic Technology
Symbols
% (percent sign) 18, 31
+ (plus sign) 18
− (minus sign) 18

A
AB_Medium tag 19, 20
AB_Short tag 19, 20
Abs function 28
absolute values 28
AC_LOCALE_CURRENCY attribute 32
AC_LOCALE_CURRENCY_FORMAT attribute 32
AC_LOCALE_CURRENCY_RADIX attribute 32
AC_LOCALE_CURRENCY_THOUSAND_SEPARATOR attribute 32
AC_LOCALE_DATE_LONG attribute 32
AC_LOCALE_DATE_SEPARATOR attribute 33
AC_LOCALE_DATE_SHORT attribute 33
AC_LOCALE_MONTHS_LONG attribute 33
AC_LOCALE_MONTHS_SHORT attribute 33
AC_LOCALE_NUM_RADIX attribute 33
AC_LOCALE_NUM_THOUSAND_SEPARATOR attribute 33
AC_LOCALE_TIME_AM_STRING attribute 33
AC_LOCALE_TIME_FORMAT attribute 33
AC_LOCALE_TIME_PM_STRING attribute 33
AC_LOCALE_TIME_SEPARATOR attribute 33
AC_LOCALE_WEEKDAYS_LONG attribute 33
AC_LOCALE_WEEKDAYS_SHORT attribute 33
accessing
system resources 49
AcGetDllEncoding() method 48
acserverconfig.xml 56
Actuate Basic
building locale maps with 11
calling C functions from 48
conversion functions for 28, 29
encoding with 44, 45, 47, 48
parsing strings with 26
restrictions for multiple code pages 49, 50
saving source code for 49, 50
administrators v
.afm files 63
AggregationLabels tag 14, 16
AIX servers
installing/uninstalling fonts 63
Albanian country code 75
alignment 74
AM string 33
AM tag 18
AM/PM symbols 18, 31
angles 28
ANSI C locales 11
application programming interfaces (APIs).
See specific Actuate API
applications
changing orientation for 70, 71
restarting 22
running 48
ar_AE constant 75
ar_BH constant 75
ar_DZ constant 75
ar_EG constant 75
ar_IQ constant 75
ar_JO constant 75
ar_KW constant 75
ar_LB constant 75
ar_LY constant 75
ar_MA constant 75
ar_OM constant 75
ar_QA constant 75
ar_SA constant 75
ar_SY constant 75
ar_TN constant 75
ar_YE constant 76
Arabic (Egypt) country code 75
Arabic country codes 75
Arabic languages 70
arctangents 28
arrays 36, 37
Asc function 45
ASCII characters 2, 3
ASCII encoding 40, 44
ASCII files. See text files
AscW function 45
Atn function 28
attributes 22, 32
Australia country code 76
Austria country code 76

B
.bas files. See source files
Basic. See Actuate Basic
BEA WebLogic Workshop 77
Belgium country code 76
Belize country code 76
bg_BG constant 76
Big Endian encoding 47
Big Fonts 3
Big5 character sets 3
binary sorts 5
binary values 5
bitmap images 74
Brazil country code 77
browsers. See web browsers
built-in operators 49
Bulgarian country code 76
bytes 46

C
C/C++ functionality 48
calendar 5
   See also dates; date formats
calling external functions 48
Canada country code 76
case mappings 6
case sensitivity 47
CCur function 28
CDate function 29
CDbl function 28
changing
   application orientation 70, 71
clock
   See also time; time formats
   formatting 31
   getting AM/PM strings for 33
   setting 18, 29
class 57
code 66
code pages
   connections and multiple 50
   converting to 48–49
   default listed 40
   developing for 3, 45, 47, 48, 49
   fonts and 3
   control contents 73
   control orientation 72, 73
default fonts 66
locale maps 21, 22
report orientation 71
char data types 48
character codes 44, 45
character mappings 2, 6
character sets
   overview 2
   single-byte vs. multi-byte 2
   supported/unsupported 43
character strings. See strings
characters
   getting code page for 45
   getting from character codes 45
   getting from code pages 46
multiple code pages and 46, 49, 52, 66
repeating in strings 45, 46
report designs and 3
sorting 5
charts 61–62, 74
   See also graphs
ChDir function 49
Chinese character sets 43, 61, 63
Chinese country codes 76
Chr function 46
Chr$ function 46
ChrW function 45
ChrW$ function 45
CInt function 28
CJK languages 58
clients 50, 52
CLng function 28
clock
   See also time; time formats
   formatting 31
   getting AM/PM strings for 33
   setting 18, 29
clusters 57
code 66
code pages
   connections and multiple 50
   converting to 48–49
   default listed 40
   developing for 3, 45, 47, 48, 49
   fonts and 3
precedence for 40
report designs and multiple 49, 50, 52, 66
strings and multiple 46
supported 63, 64
Windows platforms and 50
code points 2
Code tag 16
collation sequence 5
comma delimited data 18
command-line options 70, 71
comments 49
component references 37
configurations
database clients 50, 52
font metrics 56
connections 50
controls
changing orientation for 72, 73
positioning in designer 71, 72
problems displaying 57
repositioning contents 73
setting fonts for 66
conversions
dates 29
font metrics files 63
numbers to currency 28
report parameters 34
strings 48
copying
font metrics files 64
TrueType fonts 59
Copyright tag 11
Cos function 28
cosine 28
country codes 17, 75
creating
locale maps 11
Croatian country code 76
cs_CZ constant 76
CSng function 28
CStr function 28
currency
conversions for 28
getting decimal separators for 32
getting thousands separators for 32
specifying groupings for 17
Currency data type 28
currency formats
applying 16, 18, 31
elements described 4
getting 32
currency symbols
applying 4
format strings and 31
format tags and 16, 18
getting 32
Currency tag 18
CurrencyDecimal tag 17
CurrencyGrouping tag 17, 18
CurrencySymbol tag 16
customized_fonts.rox 57
customizing
font metrics 57
locales 22
CvDate function 29
Czech country code 76
D
da_DK constant 76
data
default fonts for 66
formatting. See formatting; formats
sorting 5
Data font option 66
data types
conversion functions for 28, 29
encoding restrictions and 49
external functions and 48
databases
configuring clients for 50, 52
connecting to 50
localization example for 35–38
Date data type 29
date format symbols 19
date format tags 16, 17
date formats
applying 16, 17, 30
constructing patterns for 19
elements described 4
getting 32, 33
locale maps and 16, 17, 19, 30
date separators 16, 17, 19, 30
Date tag 12, 16, 17, 18
DateAdd function 29
DateDiff function 29
DatePart function 29
dates
See also calendar; date formats
converting to 29
formatting 4
multi-language environments and 5
ordering elements of 17
parsing 27, 29
specifying for Japanese locales 35
DateTime tag 12, 17
DateValue function 29
day abbreviations 33
Day function 29
day of month 29
day of week 17, 29, 30, 33
DaysOfWeek tag 17
de_AT constant 76
de_CH constant 76
de_DE constant 76
de_LI constant 76
decimal places 17
decimal separators 17
currency 4, 17, 31, 32
format strings and 31
numeric values 17, 18, 26, 31, 33
Decimal tag 17
default code pages 40
default encoding 50
default fonts 61, 66
default locale 8, 11, 34
default locale ID 17
default values 34
Denmark country code 76
Design Editor. See e.Report Designer Professional
designs
Japanese locales and 34
localizing 22, 23, 24
multiple code pages and 49, 50, 52, 66
PostScript fonts and 57
specifying orientation for 70–74
user input and 3
desktop reporting 8–9
detail reports 35
developers v
DHTML reports
PostScript fonts and 57
directories
creating for TrueType fonts 59
displaying
controls 71
reports 24, 25, 71
DisplayName tag 17
DLLs 48
documentation v
DOS shell 48
dos2unix utility 63
Double data type 28
downloading search results 18
Dutch country codes 76
dynamic text controls 65
dynamic-link libraries. See DLLs
E
e.Report Designer Professional
default fonts for 66
embedding fonts and 57
running reports from 44
specifying orientation for 70
e.Reporting Server. See iServer
e.reports. See reports
Egypt country code 75
el_GR constant 76
en_AU constant 76
en_BZ constant 76
en_CA constant 76
en_GB constant 76
en_IE constant 76
en_NZ constant 76
en_US constant 76
en_ZA constant 76
encoding
backward compatibility for 45, 46
cautions for byte lengths and 46
code pages and 3
configuring databases for 50, 52
developing for 44, 45, 47, 48, 49
external C/C++ functions and 48
multi-language reports and 2
overview 40
precedence for 40, 44
restrictions for 50
setting from Component Editor 43
specifying default 50
supported character sets for 43
Encoding parameter 47
English character sets 43, 47
English country codes 76
English locales 20
Environ function 48
Environ$ function 48
environment variables 48
erdpro.jpn 34
es_ES constant 77
es_MX constant 77
Estonian country code 76
et_EE constant 76
Euro currency symbol 32
EXEC files. See executable files
executable files
embedding fonts in 56, 57
executing reports
from desktop 8–9, 44
in multi-locale environments 34
Exp function 28
exponentiation 28
expressions
conversion functions for 28, 29
parsing for 26, 27
External Basic encoding option 50
external functions 48
external source files 50
externalized fonts 56

F
fa_IR constant 76
Farsi character sets 70
Farsi country code 76
fi_FI constant 76
files
See also specific types
encoding requirements for 3
multi-language character sets for 2
opening text 37, 47
saving 49, 50
specifying encoding schemes for 47
Finish() method 73
Finland country code 77
Finnish country code 76
Fix function 28
font metrics files
converting to ISO 63
customizing 57
overview 56
renaming 57
Fontographer 62
fonts
adding to charts 61–62
applying Unicode 66–67
changing default 66
code pages for 63, 64
configuring servers for 56
developing for 61, 65
embedding 58
installing 59, 62
mapping 59
overview 3, 56–60
precedence for 57
report designs and 52
substitution for 65
uninstalling PostScript 65
unsupported 57
Fonts folder 62
fontutils command 63, 64, 65
fontutils utility 63
format fields 30
Format function 30, 31
format patterns 12, 19, 21, 30
format strings 30
format tags 16, 17, 20
Format$ function 30, 31
FormatPatterns tag 12
formats
getting 32
locale maps and 16, 17
locale precedence for 8
multiple locales and 7
overview 4
formatting data 30–31
overview 4
fr_CA constant 76
fr_CH constant 76
fr_FR constant 76
FractionDigits tag 17
fractions 28
French configurations 35
French country codes 76
full date formats 17
Full tag 19
full time formats 19
functions
calling C/C++ 48
code pages and 40
date expressions and 29
encoding with 44–47, 48
numeric expressions and 28
parsing strings with 26

G
generating
output 56
reports 24, 25
Germany country code 76
GetFactoryLocale() method 24, 25
GetLayoutOrientation() method 73
GetLocaleAttribute() method 32, 33
GetLocaleName() method 32, 33
GetPrintLocale() method 25
GetViewLocale() method 25, 26
global variables 36
graphical user interfaces 21, 22
graphics 73
graphs
See also charts
Greek country code 76
grouping separators 17
currency 18
Grouping tag 17
GUIs 21, 22

H
hash arrays 36, 37
he_IL constant 76
headers
specifying locales in 23
Hebrew configurations 70
Hebrew country code 76, 77
hexadecimal values 2
Hong Kong Supplementary characters 3, 43
Hour function 29

HP-UX systems
installing/uninstalling fonts 63
hr_HR constant 76
hu_HU constant 77
Hungarian country code 77

I
IBM-AIX systems
installing/uninstalling fonts 63
id_ID constant 77
identifiers 49
image files 74
images 73
in_ID constant 77
Indonesian country code 77
infinity symbols 18
Infinity tag 18
Information Console
localizing login pages for 17
localizing reports for 22–23
input 3
InputB function 47
InputDateMode tag 17
installing
fonts 59, 62
InstrB function 47
Int function 28
IntCurrency tag 18
Integer data type 28
interfaces. See application programming
interfaces; GUIs
Internal Basic source encoding option 50
internal source files 50
international currency symbols 18
internationalization 2
See also locales
IntFractionDigits tag 17
Ireland country code 76
IsDate function 29
iServer 21
IsNumeric function 28
ISO character sets 43
ISO currency symbols 4
it_CH constant 77
it_IT constant 77
Italian country codes 77
iw_IL constant 77

J
ja_JP constant 77
Japanese character sets 43, 61
Japanese configurations
installing fonts for 61, 63
setting up 34
Japanese country code 77
Java archives. See jar files

K
keywords 30, 49
Kill function 49
ko_KR constant 77
Korean character sets 43, 62, 63
Korean country code 77

L
Label font option 66
labels
default fonts for 66
displaying 74
referencing components in 37
language codes 17
language elements 49
See also Actuate Basic
Language variable 43
language-specific implementations. See locales
Latin character sets 58
Latvian country code 77
layout pane 71
LayoutOrientation property 72
LeftB function 47
Left$ function 47
left-to-right orientation 71, 72
LenB function 47
libraries
getting external functions from 48
localization example for 36
Liechtenstein country code 76
line breaking
for text in Actuate Basic reports 53
LineWidthPadding property 65
linguistic sorts 5
list separators 18
List tag 18
literals 49
Little Endian encoding 47
Live Report Extension. See LRX
local currency symbols 4, 18
locale codes. See locale IDs
Locale ID tag 12, 17
locale IDs 12, 17, 22
locale maps
backward compatibility for 20
changing 21, 22
creating 11
format keywords in 30
overview 11, 21–22
setting default locales in 8
XML tags described 16
Locale property 23, 24, 25
locale-dependent format 30
localemap.xml 11, 21
See also locale maps
locales
case mappings for 6
customizing 22
defaults for 40, 61
design considerations for 3, 34, 52
examples for 35–38, 72
formatting data for. See formatting data
getting information about 32, 33
getting specific attributes 32
overriding report orientation for 72
overview 2
parsing strings for 26–29
setting 22, 23–26
sorting data for 5
specifying default 8, 11, 34
supported 9
login pages
displaying locale-specific names on 17
selecting locales on 22
Long data type 28
long date formats 16, 32
Long tag 19
long time formats 19
LongYear tag 19
lowercase characters 6
–ltr option 71
LTR setting 71
lv_LV constant 77

M
Management Console
    locale-specific mappings for 21
    localizing login pages for 17
mapping
    characters 2, 6
    TrueType fonts 59
master_fonts.rox 56
medium date formats 17
Medium tag 18
medium time formats 18
methods
    See also functions; specific method
localization example for 36
Mexico country code 77
Microsoft Developer Studio 74
MidB function 47
MidB$ function 47
Minus tag 18
Minute function 29
monetary values. See currency; currency
formats
month abbreviations 33
Month function 29
month names 17, 31, 33
MonthsOfYear tag 17
multi-byte characters 2, 46
multi-language encoding 2, 3
multilingual reporting 7
multi-locale environments
    case mappings for 6
    designing reports for 3
    formatting for 4
    installing fonts for 55
    sorting data for 5
multiple code pages 43, 46, 49, 52, 66
    restrictions for 49, 50

N
NameLists tag 13
negative numbers 18
Negative tag 18
Netherlands country code 76
New Zealand country code 76
nl_BE constant 76
nl_NL constant 76
NLS_LANG, setting 51
no_NO constant 77
no_NY constant 77
Norwegian country codes 77
not a number symbols 18
NotANumber tag 18
numbers
    formatting. See numeric formats
    getting decimal separators for 33
    getting fractional part 28
    getting thousands separators for 33
locale mappings for 17, 18
    parsing 26, 28
    strings as 28
numeric formats
    applying to specific locales 31
    elements described 4

O
Oct function 28
Oct$ function 28
octal notation 28
Open function 49
Open statement 47
opening
    command prompt windows 71
    Registry Editor window 70
text files 37, 47
operators 49
orientation
    changing application window 70, 71
    changing control content 73
    changing report 71
    multi-locale environments and 3
    overview 70
setting Design Editor 71
setting programmatically 73
specifying right-to-left 70
origin 71, 72, 73
output 56, 57
    embedding fonts in 58
P

page orientation
  changing 71
  multi-locale environments and 3
  overview 70
  setting programmatically 73
PageNumbers tag 12, 17
parameters
  converting 34
  setting locales from 25
ParseDate() method 26, 27
ParseNumeric() method 26
parsing strings 26–29
patterns 12, 19, 21, 30
PDF files
  embedding fonts in 58
  PostScript fonts and 57
pdffont.map file 59
Percent tag 18
percentages 18, 31
performance 57
PFA files 63
pl_PL constant 77
Plus tag 18
PM string 33
PM tag 18
Polish country code 77
Portuguese country codes 77
Positions tag 13, 17
positive numbers 18
Positive tag 18
PostScript fonts 57, 63
  uninstalling 65
precedence
  code pages 40
  encoding 44
  fonts 57
printers
  installing fonts on 63, 64
  UNIX configurations for 65
printing
  dynamic text controls and 65
  installing PostScript fonts for 63
  setting locales for 24, 25
programmers v

programming languages
  See also Actuate Basic
programs 48
pt_BR constant 77
pt_PT constant 77
R

regional settings 8
Registry Editor
  changing application orientation 70
  opening 70
report designs
  Japanese locales and 34
  localizing 22, 23, 24
  multiple code pages and 49, 50, 66
  PostScript fonts and 57
  specifying orientation for 70–74
  user input and 3
  with multiple code pages 52
report executable files. See report object executable files
report files
  See also specific
  encoding requirements for 3
  multi-language character sets for 2
  specifying encoding schemes for 47
report object executable files
  embedding fonts in 56, 57
report servers. See i.Server; servers
ReportEncoding property 43
reports
  designing. See designing reports
  developing for 11
  displaying 22
  encoding restrictions for 50
  formatting for multiple locales 7
  locale-specific examples for 35–38
  orientation defaults for 71
  overriding page orientation for 72
  running from desktop 8–9, 44
  running in multi-locale environments 34
reserved words 49
resources 49
restarting applications 22
RightB function 47
RightB$ function 47
RightToLeft constant 72
right-to-left orientation 70
ro_RO constant 77
Romanian country code 77
.rox files. See report object executable files
--rtl option 71
ru_RU constant 77
running programs 48
running reports
  from desktop 8–9, 44
  in multi-locale environments 34
runtime code page 45, 46
runtime locales
  getting names 32
  parsing for 26, 27, 29
Russian country code 77

S
saving source files 49, 50
search results 18
Second function 29
separators
  currency 4, 17, 31, 32
  dates 18, 31, 33
  format strings and 31
  format tags for 17, 18
  numeric values 17, 18, 31, 33
  parsing and 26
  time formats 18, 33
Separators tag 13, 17, 18
sequential files 47
Serbian country code 77
serial numbers 29
servers
  configuring font metrics for 56
  installing fonts on 59, 63, 64
  printing from 25
  restarting 22
  specifying locales for 21
SetLayoutOrientation() method 73
sfdata database 35
shared libraries 48
Shell function 48
short date formats 16, 33
Short tag 18
short time formats 18
ShortHalf tag 19
ShortMonth tag 19
ShortQuarter tag 19
ShortWeek tag 19
Sign tag 18
Simplified Chinese character sets 43
Sin function 28
sine 28
Single data type 28
single-byte character sets 2
sk_SK constant 77
sl_SI constant 77
Slovak country code 77
Slovenian country code 77
Slovak country code 77
SOAP API 22
Solaris systems 63
sorting 5
source code 66
Source editor font option 66
source files 49, 50
South Africa country code 76
Spain country code 77
Spanish country codes 77
sq_AL constant 75
Sqr function 28
square roots 28
sr_YU constant 77
Str function 28
Str$ function 28
String data type 28
String$ function 46
strings
  converting to code pages 48
  encoding and 40
  entering literals in 49
  externalizing 36
  getting character codes for 45
  locale-dependent format 30
  multiple code pages and 46
  parsing 26–29
  repeating characters in 45, 46
  retrieving from arrays 37
StringW function 45
StringWS$ function 45
structure pane 71
subreports 72
substrings 47
Sun systems 63
supported locales 9
sv_FI constant 77
sv_SE constant 77
Swedish country codes 77
Switzerland country code 76, 77
Symbols tag 13, 18
system locale parameter 11
system resources 49
system settings 8
SystemLocales parameter 11

T
tags 16
Tan function 28
tangent 28
text
  adding to charts 61–62
  changing orientation for 73, 74
  default fonts for 66
  entering as literals 49
  printing from UNIX servers 65
  problems displaying 57
text controls 65
Text encoding option 50
text files
  encoding 2
  externalized strings in 36
  opening 37, 47
  writing source code to 50
text strings. See strings
TextPlacement property 74
th_TH constant 77
Thai country code 77
thousands separator 4, 26, 31, 32, 33
time 29
  See also time formats; clock
time format tags 18, 19
time formats
  applying 18, 19, 30
  constructing patterns for 20
custom locales and 22
elements described 4
getting 33
time separators 18, 33
time symbols 20

Time tag 12, 18, 19
Timespan tag 12, 19
TimeValue function 29
tr_TR constant 77
Traditional Chinese character sets 3, 43
TrueType fonts 57, 59
Turkey country code 77
Type1 PostScript fonts 57

U
UCS-2 character codes 45
UCS-2 character sets 43
UCS-2 encoding 2, 40, 45, 46
UCS-2BE encoding 47
UCS-2LE encoding 47, 49, 50
uk_UA constant 77
Ukrainian country code 77
Unicode character sets 2, 3, 43
Unicode encoding option 50
Unicode fonts 52, 66
uninstalling PostScript fonts 65
United Kingdom country code 76
United States country code 76
UNIX servers
  encoding requirements for 51
  installing fonts for 59, 62–65
  master fonts file for 56
  printing from 65
uppercase characters 6
URLs
  specifying locales in 22
  supplementary character sets and 3
UseExternalizedFonts variable 56
user input 3
user interfaces 21, 22
UTF-8 encoding 2, 43

V
values 74
  See also data
variables
  converting to numeric types 28
  localization examples for 36
Variant data type 29
Version tag 11
viewing
controls 71
reports 24, 25, 71
volumes. See Encyclopedia volumes

W
week days 17, 29, 30, 33
Weekday function 29
WindowOrientation registry key 70, 71
Windows Regional Settings 8
Windows servers
changing application orientation for 70
encoding restrictions for 50
installing fonts for 56, 62
specifying default locales for 8

X
XML tags 16, 20

Y
Year function 29
Yugoslavia country code 77

Z
zh_CN constant 76
zh_HK constant 76
zh_SG constant 76
zh_TW constant 76