One Design
One Server
One User Experience

Working in Multiple Locales

using Actuate Basic Technology

This documentation has been created for software version 11.0.5.
It is also valid for subsequent software versions as long as no new document version is shipped with the product or is published at https://knowledge.opentext.com.

Open Text Corporation
275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1
Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Document No. 170215-2-430500 February 15, 2017

Contents

About Working in Multiple Locales using Actuate Basic Technology

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.

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

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.info.gov.hk/digital21/eng/hkscs/`
`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.

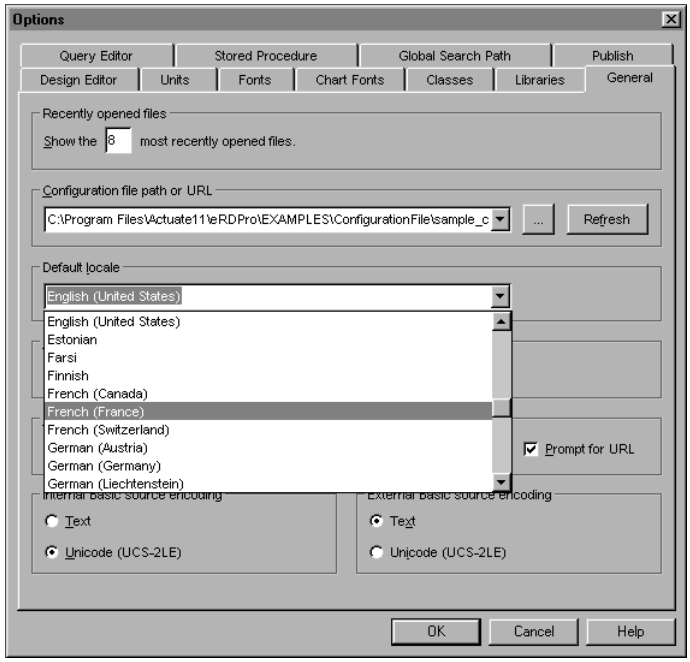

Figure 2-1 Selecting a default locale

About supported locales

Actuate Basic reports support the following locales:

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

(continues)

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

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 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>
```

```
<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>
      <PageNumbers>
         <PageNOfM>
            <Long>Page $p of $c</Long>
            <Short>P $p/$c</Short>
```

```
      </PageNOfM>
      <PageN>Page $p</PageN>
   </PageNumbers>
</FormatPatterns>
<Symbols>
   <Currency>$</Currency>
   <IntCurrency>USD</IntCurrency>
   <Plus />
   <Minus>-</Minus>
   <AM>AM</AM>
   <PM>PM</PM>
   <Infinity>∞</Infinity>
   <NotANumber>NaN</NotANumber>
   <Percent>%</Percent>
   <Separators>
      <Decimal>.</Decimal>
      <Grouping>,</Grouping>
      <CurrencyDecimal>.</CurrencyDecimal>
      <CurrencyGrouping>,</CurrencyGrouping>
      <Date>/</Date>
      <Time>:</Time>
      <List>,</List>
   </Separators>
</Symbols>
<Positions>
   <Grouping>3</Grouping>
   <CurrencyGrouping>3</CurrencyGrouping>
   <FractionDigits>2</FractionDigits>
   <IntFractionDigits>2</IntFractionDigits>
   <InputDateMode>0</InputDateMode>
   <CurrencySymbol>
      <Positive>PrecedeNoSpace</Positive>
      <Negative>PrecedeNoSpace</Negative>
   </CurrencySymbol>
   <Sign>
      <Positive>Precede</Positive>
      <Negative>Parentheses</Negative>
   </Sign>
</Positions>
<NameLists>
   <MonthsOfYear>
      <Short>
         <January>Jan</January>
         <February>Feb</February>
         <March>Mar</March>
```

```
      <April>Apr</April>
      <May>May</May>
      <June>Jun</June>
      <July>Jul</July>
      <August>Aug</August>
      <September>Sep</September>
      <October>Oct</October>
      <November>Nov</November>
      <December>Dec</December>
   </Short>
   <Full>
      <January>January</January>
      <February>February</February>
      <March>March</March>
      <April>April</April>
      <May>May</May>
      <June>June</June>
      <July>July</July>
      <August>August</August>
      <September>September</September>
      <October>October</October>
      <November>November</November>
      <December>December</December>
   </Full>
</MonthsOfYear>
<DaysOfWeek>
   <Short>
      <Sunday>Sun</Sunday>
      <Monday>Mon</Monday>
      <Tuesday>Tue</Tuesday>
      <Wednesday>Wed</Wednesday>
      <Thursday>Thu</Thursday>
      <Friday>Fri</Friday>
      <Saturday>Sat</Saturday>
   </Short>
   <Full>
      <Sunday>Sunday</Sunday>
      <Monday>Monday</Monday>
      <Tuesday>Tuesday</Tuesday>
      <Wednesday>Wednesday</Wednesday>
      <Thursday>Thursday</Thursday>
      <Friday>Friday</Friday>
      <Saturday>Saturday</Saturday>
   </Full>
</DaysOfWeek>
<AggregationLabels>
   <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>
   <Short>Overall:</Short>
```

```
            <Long>Overall:</Long>
          </OverallLabel>
        </AggregationLabels>
      </NameLists>
    </Locale>
    …
</SystemLocales>
```

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

Table 2-1 XML tags in localemap.xml

XML tag	Child tag	Description
AggregationLabels		Full and abbreviated headings and aggregation labels for the Actuate Query option, including: Average Count Maximum Minimum Overall Sum
Code		The ISO 639 language code plus the ISO 3166 country code
CurrencySymbol	Positive	Position of currency symbol for positive values: PrecedeNoSpace PrecedeWithSpaceSucceedNoSpace SucceedWithSpace
	Negative	Position of currency symbol for negative values: PrecedeNoSpace PrecedeWithSpace SucceedNoSpace SucceedWithSpace
Date	Short	Short date format
	Medium	Medium date format used by Actuate Information Console and Management Console
	Long	Long date format
	Full	Full date format
	AB_Medium	Medium date format provided for backward compatibility

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

XML tag	Child tag	Description
DateTime	Short	Short date/time format
	Medium	Medium date/time format
	Long	Long date/time format
DaysOfWeek	Short	Abbreviated day names
	Full	Full day names
DisplayName		Locale name displayed on the Information Console, and Management Console login pages
Locale ID		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 "default".
MonthsOfYear	Short	Abbreviated month names
	Full	Full month names
PageNumbers	Long	Long page number, for example Page 19 of 42
	Short	Short page number, for example P 19/42
	PageN	Page number, for example Page 19
Positions	Grouping	Number of digits to group for numbers
	CurrencyGrouping	Number of digits to group for currency
	FractionDigits	Number of digits after the decimal separator for currency
	IntFractionDigits	Number of digits after the decimal separator for numbers
	InputDateMode	Specifies the order in which the day, month, and year are entered: 0 = month, day, year 1 = day, month, year 2 = year, month, day
Separators	Decimal	Decimal separator for numbers
	Grouping	Grouping separator for numbers
	CurrencyDecimal	Decimal separator for currency
	CurrencyGrouping	Grouping separator for currency
	Date	Date separator

(continues)

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

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

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

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

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

Date symbol	Description
G	Era designator
y	Year
M	Month
d	Day of month
E	Weekday name
D	Day of year
F	Day of week in month, for example 2 represents the second Wednesday in July
w	Week of year
W	Week of month
q	Quarter of year
l (the letter el)	Half of year
'	Escape character
'' (Two single quotes)	Single quote

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

Time symbol	Description
h	Hour (1-12)
H	Hour (0-23)
m	Minute
s	Second
S	Millisecond
a	AM/PM symbol
k	Hour of day (1-24)
K	Hour of AM/PM (0-11)
z	Time zone
'	Escape character
'' (Two single quotes)	Single quote

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

Table 2-4 Sample format patterns

Format pattern	Sample result
yyyy.MM.dd G 'at' HH:mm:ss z	1996.07.01 AD at 15:08:56 PDT
EEE, MMM d, ''yy	Wed, Jul 1, ‘96
h:mm a	1:08 PM
hh 'o''clock' a, zzzz	12 o’clock PM, Pacific Daylight Time
K:mm a, z	0:00 PM, PST
yyyyy.MMMMM.dd GGG hh:mm aaa	1996.July.01 AD 02:08 PM

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 (x86)\Actuate11\iServer \servletcontainer\iportal\WEB-INF on Windows 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 (x86)\Actuate11\iServer\servletcontainer \mgmtconsole\WEB-INF on Windows 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 (x86) \Actuate11\Config on Windows 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.

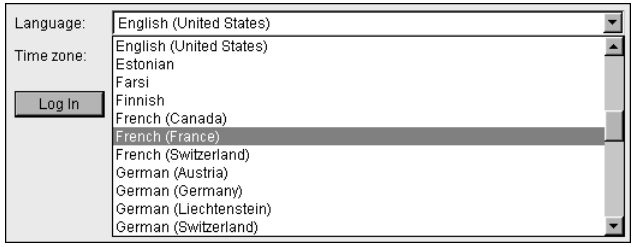

Figure 2-2 Choosing a locale on the Information Console login page

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

```
http://<web_server>/acweb/viewer/viewframeset.jsp?name=
   /detail/detail1.roi&page=1&Locale=fr_FR
```

- 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:

```
<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.

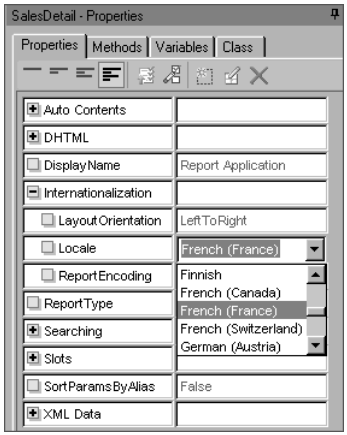

Figure 2-3 Setting the Locale property for a report

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:

```
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:

```
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( ):

   ```
   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.

4 Override the top-level report component's Start( ) method:

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

5 Override GetViewLocale( ):

```
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 run-time locale into account, while others parse strings according to the rules of the run-time 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 run-time locale into account.

Using the ParseNumeric function

To parse a numeric expression without taking the report's run-time 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 run-time 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:

```
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 run-time locale is used.

Table 2-5 lists example formats and date expressions.

Table 2-5 Examples of formats and date expressions

Format	Date expression	Description
ymd	1999-03-04	Year-month-day with four-digit year.
mdy	Nov. 12, 1972	Month-day-year with four-digit year.
mdy19	11/12/72	Month-day-year with two-digit year assumed to be in the twentieth century.
dmyp	28-3-02	Day-month-year with two-digit year that uses the default Actuate pivot date.
mdy19p25	4/1/83	Month-day-year with century and pivot. Use this format when importing data from a file.
ymdt	1999-01-28 17:15	Year-month-day with 24-hour time.
mdy12t	1/4/94 9:12 AM	Month-day-year with 12-hour time.
w?mdy12t?	Sat. May 14, 1999 12:43 PM	Month-day-year with optional weekday and time.
mdy?t12?	May 14, 1999 5/14/1999 5/14/1999 12:48 PM 12:48 pm	Month-day-year with optional date and time.

Understanding locale-dependent parsing

Many Actuate Basic functions parse strings according to the rules of the report's run-time 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 run-time locale, and are described in Table 2-6. For example:

```
CDbl("123,456") returns 123456.00 for the US locale.
CDbl("123,456") returns 123.456 for the French (France) locale.
```

Table 2-6 Functions that operate on numeric expressions

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

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 run-time locale, and are described in Table 2-7.

Table 2-7 Functions that operate on date expressions

Function	Description
CDate	Converts an expression to a Date
CVDate	Converts an expression to a Variant of VarType 7 (Date)
DateAdd	Returns a date to which a specified time interval has been added
DateDiff	Calculates and returns the time difference between two specified dates
DatePart	Returns a specified component of a given date
DateValue	Returns a date variant that represents the date of the supplied string
Day	Returns an integer between 1 and 31, inclusive, that represents the day of the month for a supplied date argument
Hour	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
IsDate	Determines whether the given argument can be converted to a date
Minute	Returns an integer from 0 to 59, inclusive, that represents the minute of the hour specified by a supplied date expression
Month	Returns an integer between 1 and 12, inclusive, that represents the month of the year for a supplied date argument
Second	Returns an integer from 0 to 59, inclusive, that represents the second of the minute specified by a supplied date expression
TimeValue	Returns a Date variant representing a time of day, based on a supplied string
Weekday	Returns an integer between 1 (for Sunday) and 7 (for Saturday) that represents the day of the week for a supplied date argument
Year	Returns an integer between 100 and 9999, inclusive, that represents the year of a supplied date argument

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 run-time 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 run-time locale and enclose it in quotation marks (").
  For example, if the report's run-time locale will be French (France), enter "3 434,2899".

If the locale code evaluates to Null or is not valid, the report's run-time 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:

```
<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 run-time 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 2-8 Locale attributes that GetLocaleAttribute( ) can return

Attribute name	Returns
AC_LOCALE_CURRENCY	Currency symbol
AC_LOCALE_CURRENCY_FORMAT	Format for currency
AC_LOCALE_CURRENCY_RADIX	Decimal separator for currency
AC_LOCALE_CURRENCY_THOUSAND_SEPARATOR	Thousands separator for currency
AC_LOCALE_DATE_LONG	Long date format
AC_LOCALE_DATE_SEPARATOR	Date separator
AC_LOCALE_DATE_SHORT	Short date format
AC_LOCALE_MONTHS_LONG	Comma-separated list of month names

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

Attribute name	Returns
AC_LOCALE_MONTHS_SHORT	Comma-separated list of month abbreviations
AC_LOCALE_NUM_RADIX	Decimal separator for numbers
AC_LOCALE_NUM_THOUSAND_SEPARATOR	Thousands separator for numbers
AC_LOCALE_TIME_AM_STRING	AM string
AC_LOCALE_TIME_FORMAT	Time format
AC_LOCALE_TIME_PM_STRING	PM string
AC_LOCALE_TIME_SEPARATOR	Time separator
AC_LOCALE_WEEKDAYS_LONG	Comma-separated list of day names
AC_LOCALE_WEEKDAYS_SHORT	Comma-separated list of day abbreviations

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 run-time locale:

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

If the run-time 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 run-time locale. If the run-time locale is US English, Start date appears in the report as 11/30/2002. If the run-time 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 run-time locale. If the run-time locale is French (France), Start date appears in the report as 30/11/2002. If the run-time 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, <eRDPro_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.

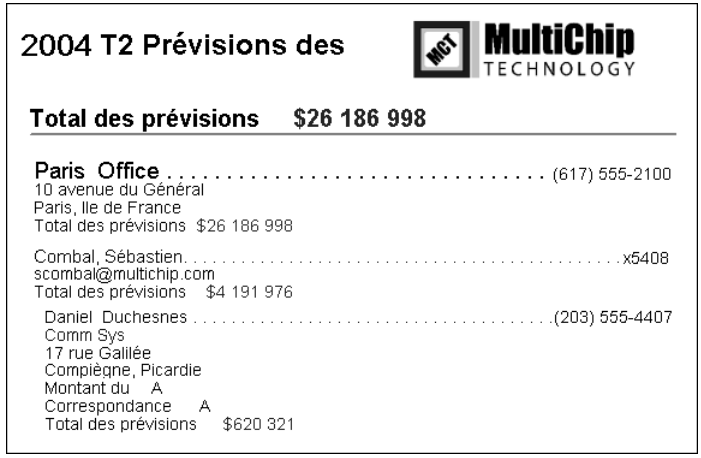

Figure 2-4 Example of a localized report

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.

itemcode	description	pricequote	quantity	category	orderID
MP1632	Contrôleur embarqué programmable 32 bits	210	49	Contrôleur	1075
MR3240	4 x 32 Mo de Ram dynamique	31	50	Ram dynamique	1075
MRL0480	4 x 8 Mo de Ram dynamique, 3,3 volts	18	49	Ram dynamique	1075
MRL3240	4 x 32 Mo de Ram dynamique, 3,3 volts	61	49	Ram dynamique	1075
MS0880	8 x 8 Mo de Ram statique	52	49	Ram statique	1075

Figure 2-5 Example of a localized database

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::LabelControl11, MultiChip Technology
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:

```
Declare
   Global labelDictionary As MyDictionary
End Declare
```

```
Class MyDictionary
   Dim KeyLen(9)      As Integer
   Dim Key0( )        As String
   DimKey1( )         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:

```
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:

```
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.

Chapter

3

Understanding report encoding

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

Locale	Code page
Albanian	windows-1250
Arabic (Algeria)	windows-1256
Arabic (Bahrain)	windows-1256
Arabic (Egypt)	windows-1256
Arabic (Iraq)	windows-1256
Arabic (Jordan)	windows-1256
Arabic (Kuwait)	windows-1256
Arabic (Lebanon)	windows-1256
Arabic (Libya)	windows-1256
Arabic (Morocco)	windows-1256

Table 3-1 Default code pages for supported locales (continued)

Locale	Code page
Arabic (Oman)	windows-1256
Arabic (Qatar)	windows-1256
Arabic (Saudi Arabia)	windows-1256
Arabic (Syria)	windows-1256
Arabic (Tunisia)	windows-1256
Arabic (U.A.E.)	windows-1256
Arabic (Yemen)	windows-1256
Bulgarian	windows-1251
Chinese (Hong Kong SAR)	windows-950 (includes Hong Kong Supplementary Character Set)
Chinese (PRC)	windows-936
Chinese (Singapore)	windows-936
Chinese (Taiwan)	windows-950 (includes Hong Kong Supplementary Character Set)
Croatian	windows-1250
Czech	windows-1250
Danish (Denmark)	windows-1252
Dutch (Belgium)	windows-1252
Dutch (Netherlands)	windows-1252
English (Australia)	windows-1252
English (Belize)	windows-1252
English (Canada)	windows-1252
English (Ireland)	windows-1252
English (New Zealand)	windows-1252
English (South Africa)	windows-1252
English (United Kingdom)	windows-1252
English (United States)	windows-1252
Estonian	windows-1257
Farsi	windows-1256

(continues)

Table 3-1 Default code pages for supported locales (continued)

Locale	Code page
Finnish	windows-1252
French (Canada)	windows-1252
French (France)	windows-1252
French (Switzerland)	windows-1252
German (Austria)	windows-1252
German (Germany)	windows-1252
German (Liechtenstein)	windows-1252
German (Switzerland)	windows-1252
Greek	windows-1253
Hebrew	windows-1255
Hungarian	windows-1250
Indonesian	windows-1252
Italian (Italy)	windows-1252
Italian (Switzerland)	windows-1252
Japanese	windows-932
Korean	windows-949
Latvian	windows-1257
Norwegian (Bokmal)	windows-1252
Norwegian (Nynorsk)	windows-1252
Polish	windows-1250
Portuguese (Brazil)	windows-1252
Portuguese (Portugal)	windows-1252
Romanian	windows-1250
Russian	windows-1251
Serbian (Latin) (Yugoslavia)	windows-1251
Slovak	windows-1250
Slovenian	windows-1250
Spanish (Mexico)	windows-1252
Spanish (Spain)	windows-1252
Swedish (Finland)	windows-1252
Swedish (Sweden)	windows-1252

Table 3-1 Default code pages for supported locales (continued)

Locale	Code page
Thai	windows-874
Turkish (Turkey)	windows-1254
Ukrainian (Ukraine)	windows-1251

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 3-2 Supported encodings

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

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.

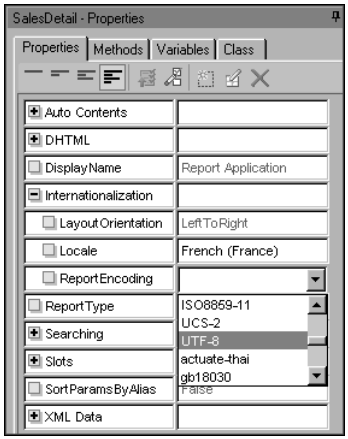

Figure 3-1 Setting the ReportEncoding property

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 Actate 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

The ChrW 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 run-time code page. The run-time 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 run-time code page for the first character in the string. The first character in the string must be present in the run-time code page. For example, if the first character in the string is a Japanese character and the run-time 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 run-time code page and return the character. For example:

```
Chr$(65) returns A
Chr(227) returns ã if run-time code page is Western European
Chr(227) returns γ if run-time code page is Greek
```

Using the String$ function

The String$ function takes a numeric expression and a character code in the run-time 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 γγγγγ if run-time code page is Greek
String$(5,227) returns ããããã if run-time 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 run-time code page. The run-time 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 run-time 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

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

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 run-time encoding is used.

The file name must use the operating system code page.

For example:

```
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
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.

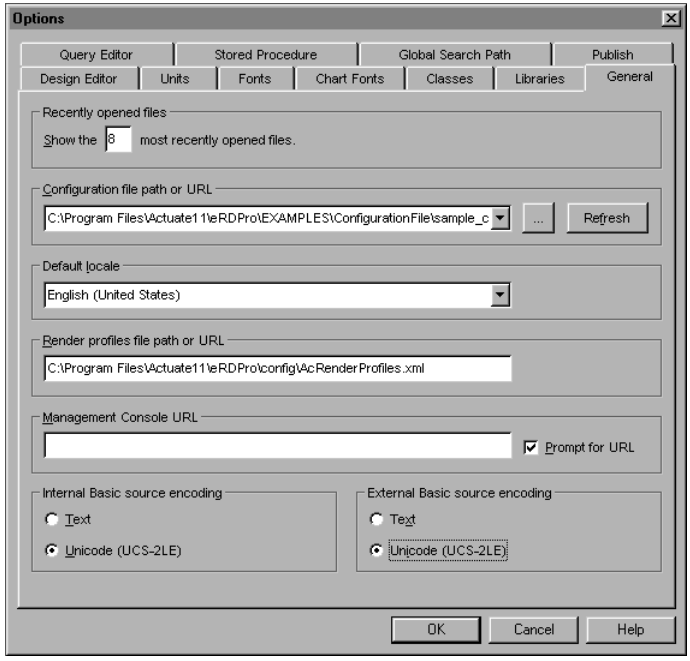

Figure 3-2 Selecting a source encoding to use

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 and Solaris platforms.

Table 3-4 Supported values for LC_ALL

AIX	Solaris
ibm-1370	ibm-1383
ibm-1383	ibm-1386
ibm-1386	ibm-33722
ibm-33722	ibm-915
ibm-850	ibm-943_P14A-2000
ibm-943_P14A-2000	ibm-970
ibm-970	LATIN_1
LATIN_1	UTF8
UTF8	

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 \xff0c \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:

```
TurnOnAsianLineBreakingRule = 1
DoNotBegin = ",.?!)]}>\x3001\x3002\xff0c\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 (x86)\Actuate11\iServer\etc\master_fonts.rox on Windows 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.
    2. In Properties settings, choose View Service➛PDF Generation➛PDF Font Directory.
    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
```

5 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.

6 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 uppercase 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 uppercase 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 4-1 Default fonts by locale

Locale	Default fonts
Chinese (Simplified)	simsun.ttc
Chinese (Traditional)	mingliu.ttc
Japanese	msgothic.ttc msmincho.ttc
Korean	gulim.ttc batang.ttc
Other	arial.ttf arialbd.ttf arialbi.ttf ariali.ttf times.ttf timesbd.ttf timesbi.ttf timesi.ttf

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.
7. In Fonts, choose File➛Close.

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.

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 -1 <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

     <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

     <Windows_font_name> 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.
2. In e.Report Designer Professional, choose Tools➤Options.
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.

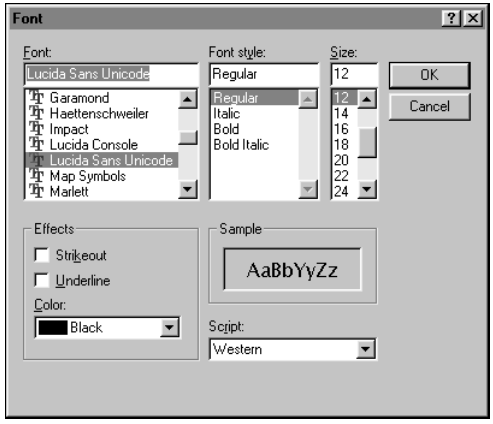

Figure 4-1 Specifying font formatting

6 In Options, choose OK.

Chapter

5

Designing reports with right-to-left orientation

This chapter contains the following topics:

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 operating systems. You can change the orientation on other Windows 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.

8 Choose OK.

9 Choose Registry➛Exit.

10 Start e.Report Designer Professional.

e.Report Designer Professional starts with right-to-left orientation.

Use a command line option:

1 On the task bar, choose Start➛Programs➛Command Prompt.

The Command Prompt window appears.

2 Type:

```
cd <eRDPro_HOME>\bin
```

3 Type:

```
erdpro -rtl
```

e.Report Designer Professional starts with right-to-left orientation.

To change back to left-to-right orientation, set the WindowOrientation registry key to LTR, or use the -ltr command line option. For more information about command line options, see *Developing Reports using e.Report Designer Professional*.

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.

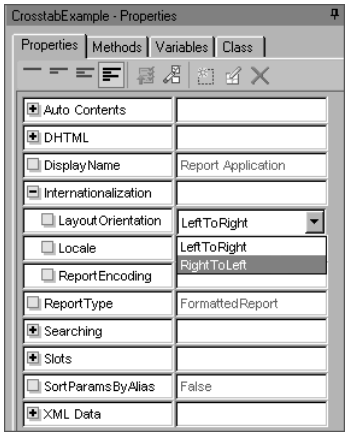

Figure 5-1 Selecting right-to-left orientation

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.

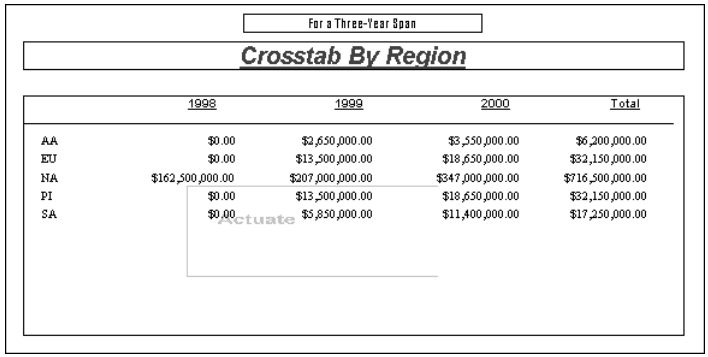

Figure 5-2 Example of a report with left-to-right orientation

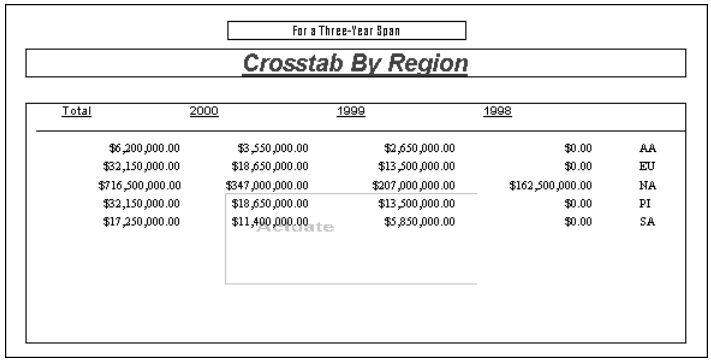

Figure 5-3 Example of a report with right-to-left orientation

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:

```
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( ).SetLayoutOrientaiton(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.

Appendix

A

Locale codes

Use the locale codes in Table A-1 to identify locales in Actuate Basic code.

Table A-1 Locale codes for Actuate Basic

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

(continues)

Table A-1 Locale codes for Actuate Basic (continued)

Locale	Code
Arabic (Yemen)	ar_YE
Bulgarian	bg_BG
Chinese (Hong Kong SAR)	zh_HK
Chinese (PRC)	zh_CN
Chinese (Singapore)	zh_SG
Chinese (Taiwan)	zh_TW
Croatian	hr_HR
Czech	cs_CZ
Danish (Denmark)	da_DK
Dutch (Belgium)	nl_BE
Dutch (Netherlands)	nl_NL
English (Australia)	en_AU
English (Belize)	en_BZ
English (Canada)	en_CA
English (Ireland)	en_IE
English (New Zealand)	en_NZ
English (South Africa)	en_ZA
English (United Kingdom)	en_GB
English (United States)	en_US
Estonian	et_EE
Farsi	fa_IR
Finnish	fi_FI
French (Canada)	fr_CA
French (France)	fr_FR
French (Switzerland)	fr_CH
German (Austria)	de_AT
German (Germany)	de_DE
German (Liechtenstein)	de_LI
German (Switzerland)	de_CH
Greek	el_GR
Hebrew	he_IL

Table A-1 Locale codes for Actuate Basic (continued)

Locale	Code
Hebrew	iw_IL (Use only with Actuate iServer Components for BEA WebLogic Workshop.)
Hungarian	hu_HU
Indonesian	id_ID
Indonesian	in_ID (Use only with Actuate iServer Components for BEA WebLogic Workshop.)
Italian (Italy)	it_IT
Italian (Switzerland)	it_CH
Japanese	ja_JP
Korean	ko_KR
Latvian	lv_LV
Norwegian (Bokmal)	no_NO
Norwegian (Nynorsk)	no_NY
Polish	pl_PL
Portuguese (Brazil)	pt_BR
Portuguese (Portugal)	pt_PT
Romanian	ro_RO
Russian	ru_RU
Serbian (Latin) (Yugoslavia)	sr_YU
Slovak	sk_SK
Slovenian	sl_SI
Spanish (Mexico)	es_MX
Spanish (Spain)	es_ES
Swedish (Finland)	sv_FI
Swedish (Sweden)	sv_SE
Thai	th_TH
Turkish (Turkey)	tr_TR
Ukrainian (Ukraine)	uk_UA

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z