Java String Methods in Print Templates

Print templates in Millennium/Sierra provide a great deal of flexibility in printed output and notices, both in terms of layout and in the ability to combine and manipulate data. Where Static Text data elements contain only unchanging text labels, Text Fields can contain expressions ranging from simple to very complex. The purpose of this document is to illustrate some of the techniques that can be used in Text Fields.

Simple Field

A Text Field data element may simply contain a single field. This example from a spine label template would print the contents of subfield f of the call number (i.e. the pre-stamp). Note that only fields that have been defined by Innovative for the particular template can be used. For a list of data libraries, see IGR #107515 or the Sierra documentation at Sierra Reference > Print Template Data Libraries.

Concatenation

A Text Field may contain only one expression. Use the plus sign (+) to concatenate fields into one string, or to join fields with literal text. Literal text must be in quotes. Line breaks and spaces between elements do not print, but may be added to an expression for clarity. Use "\n" to print carriage returns.

Java String Methods

Java string methods can be used to manipulate string values and develop more complex expressions. In print templates, string methods have this structure:  

[field name] . [method name] ([list of arguments])

Depending on the string method, the result may be another string, a Boolean value (true/false), or an integer:

- `$F{bib245}.toUpperCase()` changes “Call of the wild” to “CALL OF THE WILD”
- `$F{itemFix79}.equals("gcrf ")` returns true if the Item Location is “gcrf”; otherwise false
- `$F{v_p_name}.indexOf("","`)` returns true if the Patron name contains “,” (comma-space)
- `$F{bib245}.length()` For the title “Call of the wild”, this string method returns the integer 16 — the length of the string
- `$F{v_p_name}.indexOf("","")` For the Patron name “Doe, John”, this returns 3 — the position of the first occurring comma. (Counting starts at 0.)

Note that string method names are case-sensitive: `indexOf` is a valid method; `indexOf` is not.

For a summary of Java string methods, see:  [http://docs.oracle.com/javase/6/docs/api/java/lang/String.html](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html)

Conditional statements

Conditional statements have an “if-then-else” structure, allowing you to control printing based on your data. The syntax is:  

[Boolean expression] ? [expression to perform if true] : [expression to perform if false]

This means: If the Item Volume field is empty, then print nothing; else print a carriage return.
This example shows a nested conditional. The expression means: If the Item Location is “gcrf”, print “REF”; otherwise, if the Location is “hmrf”, print “MSS REF; otherwise, print nothing.

Substrings

The `.substring` method is used to extract part of a field value. When one integer argument is given, the method extracts everything from that character position to the end. (Counting starts at 0.) Two integer arguments separated by a comma specify both the starting character position and the character position immediately after the last desired character.

This results in that part of Item Barcode starting at character position 10. For a 14-digit barcode, it would print the last 4 characters.

This expression would extract the string “41145” from the patron barcode “20006000411457”. First, the `.substring` method pulls out characters 6-12 (counting starts at 0), resulting in “0041145”. Then the `.replaceAll` method replaces any and all leading 0’s with the empty string “”, that is, it removes them. Note that “^0*” is a regular expression, indicating any number of 0’s occurring at the beginning of the string.

A Complex Example

This complex expression converts a Patron Name to an abbreviated form, such as for privacy on a hold slip. It uses nested conditional statements, and substring methods in which the start and end points are themselves string methods. It accounts for short last names, no comma, and a varying number of spaces after the comma.

Translation:

If the Patron Name field contains a comma,
  then: If the comma occurs before the character position 3 (i.e., if last name is shorter than 3 characters)
    then: Select the Name field up to the comma;
    else: Select the first 3 characters of the Name field
  Concatenate “/”
  Select the 4 characters following the comma, trim any leading spaces, then select the first character of what remains (this accounts for 0-3 spaces between the comma and the first name);
else: (Name field does not contain comma) Select the first 4 characters of the Name field
Convert the entire result of the above to upper case

Examples:

Jackson, Richard V. prints as JAC/R
Au, Janice prints as AU/J
Test patron prints as TEST
A Summary of Some Useful String Methods

(See http://docs.oracle.com/javase/6/docs/api/java/lang/String.html for a more complete summary.)

**Methods returning strings:** The following Java string methods return string results, which can then be printed or be passed to further string methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toUpperCase()</code></td>
<td>none</td>
<td>Converts all lower case characters in the string to upper case.</td>
</tr>
<tr>
<td><code>trim()</code></td>
<td>none</td>
<td>Returns a copy of the string, with leading and trailing spaces removed.</td>
</tr>
</tbody>
</table>
| `substring()`   | integer [, integer] | 1st argument: the character position for the start of the substring  
|                 |                    | 2nd argument: the character position immediately following the desired end of the substring (may be omitted)  
|                 |                    | *Note: Counting of character positions always starts at 0*                   |
|                 |                    | Description: Returns a substring of the given string, from the character position of the 1st argument up to (but not including) the character position of the 2nd argument (or, if 2nd argument omitted, to the end of the string) |
|                 |                    | Example: `$F{Patron_Barcode}.substring( 8 )`  
|                 |                    | *For the barcode “2006000411457”, this would print “411457”*               |
|                 |                    | Example: `$F{Patron_Name}.substring( 0 , 4 )`  
|                 |                    | *For the patron name “Smith, Jane,” this would return “Smit”*               |
| `replace(string , string)` | 1st argument: the string to match within the given string  
| | 2nd argument: the string with which to replace it | Description: Replaces all substrings that match the string in the first argument with the literal string specified in the second argument  |
|                 |                    | Example: `$F{callEntire}.replace( " ", "\n" )`  
|                 |                    | *Converts the call number “DVD SCI-FI 123” to “DVD\nSCI-FI\n123”. (Note: “\n” generates a carriage return.)* |
| `replaceAll( string (regex) , string)` | Arguments: same as for replace, but the 1st argument can be a regular expression | Description: Replaces all substrings that match the regular expression in the first argument with the literal string specified in the second argument.  |
|                 |                    | Example: `$F{Patron_Name}.replaceAll( "●●" , "●" )`  
|                 |                    | *[● signifies a space]*  
|                 |                    | Replaces occurrences of 2 or more consecutive space characters with a single space. For example, it would change “Jackson, Richard” to “Jackson, Richard”.* |
### Methods returning Boolean values
The following string methods return Boolean values (true or false). These should not be printed, but can be used as the first part of an “if-then-else” conditional statement.

<table>
<thead>
<tr>
<th>Method</th>
<th>Argument</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>equals()</code></td>
<td>the string to be compared to the given string</td>
<td>Returns true if the given string is identical to the string in the argument</td>
<td><code>$F{itemFix58c}.equals(&quot;c.1&quot;) ? &quot;&quot; : $F{itemFix58c}</code>&lt;br&gt; <em>If the <code>itemFix58c</code> field (copy number prefixed with “c.”) equals “c.1”, do nothing, otherwise, print the copy no.</em></td>
</tr>
</tbody>
</table>
| `startsWith()` | string to be compared to the beginning of the given string | Returns true if the given string begins with the string in the argument                                                 | `$F{callEntire}.startsWith("M") ? "Music Room" + \\
+  "\n" : [...]`<br> *If the call number starts with “M”, print “Music Room” plus a carriage return* |
| `endsWith()`   | string to be compared to the end of the given string | Returns true if the given string ends with the string in the argument                                                   | `$F{itemFix79}.endsWith("o") ? "OVERSIZE" + \\
+  "\n" : [...]`<br> *If the Item Location code ends with “o”, print “OVERSIZE” plus a carriage return (e.g., as a call no. pre-stamp)* |
| `contains()`   | string to be sought within the given string | Returns true if the given string contains the string specified in the argument                                          | `$F{Patron_Name}.contains(" , ") ? [...] : [...]`<br> *Tests whether the Patron Name field contains the character sequence comma-space* |
| `isEmpty()`    | none | Returns true if the given string contains no data (has a length of 0)                                                     | `$F{itemv}.isEmpty() ? "" : $F{itemv} + \\
+  "\n"`<br> *If the Item Volume field does not exist, print nothing; otherwise, print the Item Volume and a carriage return (Note: Avoid using the `isEmpty` method with fields based on fixed-length fields in Millennium/Sierra)* |
| `matches()`    | regular expression to be matched against the given string | Returns true if the given string matches the regular expression in the argument                                         | `$F{callEntire}.matches("^\[A-Z\]{1,3}\[0-9\]{1,4}\.*") ? [...] : [...]`<br> *Tests whether the call number begins with a pattern of 1-3 upper case letters followed by 1-4 numbers* |
A Summary of Some Useful String Methods (continued)

Methods returning integers: The following methods return integer values. These should not be printed, but may be used as part of more complex expressions. For example, a string method returning an integer can be used as an argument of any string method that calls for integer arguments, such as the substring method. Integer results can be combined with arithmetic operations such as addition (+) and subtraction (-) to yield new integers. They can also be used with comparison operators such as equals (=), greater than (>), and less than (<); the results of such comparisons are Boolean values that can be used in conditional statements.

<table>
<thead>
<tr>
<th>Method</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length()</td>
<td>none</td>
<td>Returns the length of the given string</td>
</tr>
<tr>
<td>Example:</td>
<td>$F{Item_Barcode}.trim().length() &lt; 14 ? [...] : [...]</td>
<td>The length method returns the length of the barcode (after any leading or trailing spaces have been trimmed). This result is compared with the number 14, returning a Boolean value of true if the length if less than that.</td>
</tr>
<tr>
<td>Example:</td>
<td>$F{Patron_Barcode}.substring( $F{Patron_Barcode}.length() - 6 )</td>
<td>6 is subtracted from the length of the patron barcode. This result is used as the argument in a substring method, enabling the last 6 characters of the barcode to be printed, even when the length of the barcode varies. (Note: An error will result in this case if the barcode is shorter than 6 characters.)</td>
</tr>
<tr>
<td>indexOf( string [, int] )</td>
<td>1st argument: the character or string to be sought within the given string 2nd argument (optional): the character position at which to begin the search</td>
<td>Returns the index (character position) within the given string of the first occurrence of the character or string specified in the first argument. The second argument, if given, specifies the character position (an integer) at which to begin the search.</td>
</tr>
<tr>
<td>Example:</td>
<td>$F{Patron_Name}.substring( 0, $F{Patron_Name}.indexOf(&quot;,&quot; ) )</td>
<td>The indexOf method returns the character position of the first comma in the Patron Name field. This is then used as the second argument in a substring method, which results in the patron’s name from the beginning up to (but not including) the first comma. For the name “Jackson, Richard, Jr.”, this would print “Jackson”. (Note: If the patron name does not contain a comma, the indexOf method will return -1. This in turn will cause an error in the substring method, since the end position is now less than the start position. Therefore, this expression should be placed within a conditional statement that tests for the presence of a comma: $F{Patron_Name}.contains(&quot;,&quot; ) ? [the above expression] : [...]</td>
</tr>
<tr>
<td>lastIndexOf( string [, int] )</td>
<td>1st argument: the character or string to be sought within the given string 2nd argument (optional): the position back from which to begin the search</td>
<td>Returns the index (character position) within the given string of the last occurrence of the character or string specified in the first argument. The second argument, if given, specifies the character position (an integer) at which to begin the search (searching backward).</td>
</tr>
<tr>
<td>Example:</td>
<td>$F{callEntire}.substring( $F{callEntire}.lastIndexOf(&quot; &quot;) + 1 )</td>
<td>The lastIndexOf method returns the character position of the last space character in the Patron Name field. 1 is then added to this, enabling the substring method to select the last “word” of the call number. For the call number “Z694.W425 2012”, this expression would return “2012”</td>
</tr>
</tbody>
</table>
**Formatting Dates**

To insert the system date (today’s date) into any print template, add this expression to a Text Field:

```
new java.util.Date()
```

To format this date in iReport, right-click on the field, choose “Field pattern,” and select the format you want from the list, as shown at right. This sets the field pattern attribute to (in this example) “MMMMM dd, yyyy” to print the date as “April 23, 2014.” (If you don’t set a field pattern, the default pattern is “MM/dd/yyyy” (“04/23/2014”).

However, date fields that are pulled from Millennium/Sierra records, such as due date or checked out date:

```
$F{Item_Due_Date} $F{Item_Checked_Out_Date}
```

cannot be formatted using this method. That is because such fields are string fields, stored as text, not as dates. On its own, a date field will typically be printed in the format “mm-dd-yyyy” (e.g., 04-23-14), although this may vary slightly depending on how your system is set up. (Other possible patterns are “mm-dd-yyyy”, “dd-mm-yyyy”, or “dd-mm-yyyy”.)

Nevertheless, it is possible to change the format of a due date in your print template, and a Text Field expression for doing so with a due date is shown below. It uses “SimpleDateFormat,” first to parse the date into a true date field, then to format that date in whatever pattern you want:

```
new SimpleDateFormat("MMMM d, yyyy").format(new SimpleDateFormat("MM-dd-yyyy").parse($F{Item_Due_Date}))
```

This expression may be easier to interpret if you analyze it from right to left. First, the Item Due Date is parsed to create a true date field. The pattern "MM-dd-yyyy" tells the parser how to interpret the string of characters output by the field “$F{Item_Due_Date}”; it should match the way this field normally prints. This date field is then converted back into a printable string in whatever format you choose. In this example, the pattern "MMMM d, yyyy" would cause a due date of "07-08-14" to be printed as “July 8, 2014”. (Note that in iReport, upper case “M” is used for months; lower case “m” is used for minutes.)

Many other formats are possible:

- "EEE dd MMMM yyyy" would print the date as: “Tue 08 July 2014”
- "MMMMM d, 'yy" would print the date as: “July 8, '14”
- "M/d/yyyy" would print the date as: “7/8/2014”
- "dd-MMM-yyyy" would print the date as: “08-Jul-2014”
- "MMMMM d (EEEE)" would print the date as: “July 8 (Tuesday)”

See [http://docs.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html](http://docs.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html), particularly the table under “Date and Time Patterns,” for more information on formatting dates.