The MiscMergeKit

Library: libMiscMerge.a (also requires libMiscKit.a)

Header File Directory: /LocalDeveloper/Headers/misckit

Import: misckit/miscmerge.h

Introduction

The MiscMergeKit implements a generic engine which may be used to combine data records and template files. A template file is defined using a ªmerge languageº (described below) and contains ªblanksº. The blanks are filled in with data taken from a data record, which is a MiscDictionary. The name of the blank is used as a key into the data record. Other merge commands allow conditional text, comments, and other features. The

engine itself is meant to be embedded in a program and is flexible to be used for many purposes:

- Document merge capability (inside word processors or other user apps).
- Programmatic generation of code in C, Objective C, or other languages.
- Programmatic generation of documentation.
- Programmatic generation of Web pages.
- Anything else you can imagine¼

This kit is the result of my (Don Yacktman) experience with this type of code. I have implemented these types of engines before andÐas part of the learning processÐhave repeatedly refined my designs. I am fairly satisfied with this incarnation and have therefore made it available in the MiscKit. My previous projects using this type of technology include school projects (versions of yacc, lex, and an optimizing compiler), graduate research (a CAD tool that extends VHDL and generates VHDL code based upon the extensions' parameters), and a user app that generates personalized E-mail spams. I am currently working on a cgi-bin program that generates web pages on the fly amongst other things. Some of the future MiscKit demo apps/programmer aids I plan to write will rely heavily upon this technology.

MiscMergeKit Classes and Protocols

Below is a diagram showing the heirarchy of MiscMergeKit classes and the other resorces provided by the kit.

MiscMergeKit.eps ¬

Figure 1. MiscMergeKit classes

Basic merging

The MiscMergeEngine class is the starting point for a basic, one-shot merge. All you have to do is:

1. Initialize a MiscMergeTemplate instance for the template you wish to use
2. Load a MiscDictionary with the necessary key/value pairs
3. Pass the MiscMergeTemplate and MiscDictionary instances to the MiscMergeEngine
4. Start the merge
5. Do something with the results, a MiscString instance, that the MiscMergeEngine returns to you.

If there is more than one data record, then a MiscMergeDriver instance should be used to perform the merge. You pass it an initialized MiscMergeTemplate and a List object filled with a MiscDictionary for each merge. You get back a new List object filled with MiscStrings, the results of the individual merges. The MiscStrings are in the same order as the MiscDictionaries on the input side, so it is easy to correlate the output to the input.

A simple example of using a MiscMergeDriver, named MergeTest, is in the MiscKit Examples area. To generate the List object populated with MiscDictionaries, the MiscRecordParser class is used. This example is quite simple, as demonstratedÐit only takes about 15 lines of code to load the template and records from a file, merge them, and print the output to stdout!

Advanced features

If you need to actually modify the merge language, it is possible to do so. The API to the MiscMergeCommand class is provided to allow you to add commands or override existing commands. Since all new commands are subclasses of MiscMergeCommand, there is a rich set of functions availble to aid in parsing and writing the

behavior of new commands via inheritance. The MiscIfStack object, MiscMergeCondCallback protocol, and MISC_Merge_Cond_Op type are available to aid in creating commands that use an if/else/endif semantic or some other type of conditional. The MiscMergeDriver protocol is provided to aid in creating custom MiscMergeDriver style classes if the provided class is inadequate for any reason. The current driver is flexible with choice of engine, so in most applications it works well. If you are interested in any of these features, you should consult the applicable class documentation, which describes the details of these features. Be sure to read the class documentation since there are many more options and hooks to allow customization of the engine to fit your needs.

MiscMergeKit merge language

The merge language is quite simple. Two delimiters are used to tell the template parser when a merge command begins and ends. By default, commands should be enclosed in pairs of ª«º and ª»º, but you can change this to brackets (ª{º and ª}º) or any other pair of characters. The first word after the opening delimiter is used as the command name. It is used to look up a command class. If the class is not found, then the word is taken to be a key into the MiscDictionary of merge variables and the merge command will substitute the value found in the dictionary for the merge command on the output. So the simplest example would be:

Template:

```
This is a sample template for «name».
```

Dictionary:

```
name = "Don Yacktman"
```

Since there is no ªnameº command, the value of the key ªnameº will be substituted into the output, to give the following output:

```
This is a sample template for Don Yacktman.
```

When searching for keys, the MiscMergeEngine attempts to resolve the values as far as it can. So, for example, one value in the dictionary could be another key in the dictionaryÐcausing an indirection to take place. If keys aren't found in the merge dictionary, a global dictionary is consulted. If still not found, then the key is returned as a literal value to be inserted into the template. Here are some examples:

Template:

```
This is a sample template for «name».
```

Dictionary:

```
<empty>
```

Output:

```
This is a sample template for name.
```

Dictionary:

```
name = "fullName"
fullName = "Don Yacktman"
```

Output:

```
This is a sample template for Don Yacktman.
```

If an actual command is found, then that command will be executed. What does or doesn't get placed into the output of the merge depends upon the command used. In fact, the syntax of the command itself depends upon the command. The command descriptions below detail what parameters (if any) are expected to follow a particular command. Obviously, any commands you create will have the syntax you specify. If a command contains nested merge commands, then it will not be parsed until merge time. In fact, a special MergeEngine will be created to merge the command and *then* it will be parsed. Although this considerably slows performance, it allows the merge commands to *change* depending upon the input data! This ability turns out to be extremely useful¼but it can be dangerous if you aren't careful. (I have used this feature primarily with custom commands I have created, where the parameters come from values in the MiscDictionary for the merge.)

Finally, note that the command keywords are case insensitive. Thus, ªCopyº, ªcopyº, and ªCOPYº all refer to the same command as far as the MiscMergeTemplate parsing machinery is concerned. Many of the commands are similar to the merging commands used by the WriteNow.app distributed with NEXTSTEP 2.1 and earlier since that language was used as an example when this language was created. This language is both richer and extensible, however.

copy

Copies all the text following the ªcopyº keyword to the merge output. Plain text in the template is turned into ªcopyº commands internally by the MiscMergeTemplate parsing mechanism. You'll probably never use this as a commandÐbut should remember to avoid using the word ªcopyº as a key in the merge dictionary, since that would invoke this command!

comment

This command is a no-op. Anything following the ªcommentºº keyword is ignored and discarded. Nothing is inserted in the merge output.

date

Places the date in the merge output. Right now, this is of the form ªMonth day, yearº such as ªJuly 21, 1995º. In the future, any text following the ªdateº keyword will be used as an argument to strftime() to format the date and time.

field

This uses the text following the keyword ªfieldº as a key in the merge dictionary and places the value found in the merge output. This is the long form and can be used to get around problems with keys that have the same names as commands. For example, the command in the example above could have been written ª«field name»º instead of ª«name»º. As another example, ª«copy»º won't work for retrieving the value for ªcopyº from the merge dictionary, but ª«field copy»º *will* work.

if, else, endif

These three commands allow conditional text output with a merge. Here would be a way to print out a different string of text based upon the value of the key ªsalaryº:

Template:

```
Congratulations!  You qualify for our offer for a free
Visa «if salary > 35000»Gold«else»Classic«endif» card!
```

Dictionary:

```
salary = "20000"
```

Output:

```
Congratulations!  You qualify for our offer for a free
Visa Classic card!
```

Dictionary:

```
salary = "40000"
```

Output:

```
Congratulations!  You qualify for our offer for a free
```

```
Visa Gold card!
```

The conditionals accepted by the **if** command match the C operators: <, >, >=, <=, ==, !=. Also accepted are: =, <>, ><, =>, =<. The value on either side of the operator is tried as a merge dictionary key first. If no key exists, then its literal value is used. The comparison will be numeric if the strings begin with a number (such as the 35000, 20000, and 40000 in the above example). Complex ªandº, ªorº, and ªnotº expressions are **not** allowed at this time. Similar effects can be obtained by nesting **if** statements, however, and this is the recommended procedure. Case types of statements may also be simulated this way. Look at the MiscMerge example app's template for a demonstration of nested **if** statements.

identify

This allows a value for a key to be determined. For example, ª«identify name = f0»º will make the key ªnameº return ªf0º. As part of the field resolutin performed by the merge engine, the key ªf0º will then be searched for. If not found, the text ªf0º would be returned, otherwise the value of the key ªf0º would be returned. This allows aliases for key names to be created as well as simple setting the values of keys. Note that the statement requires an ª=º or ª==º operator fo it to be parsed correctly.

set

This is the same as the **identify** command except that the value is stored in a global dictionary used by the merge engine. This allows you to set defaults for all mergesÐif the key is not found in the local dictionary, then the value found in the global dictionary will be returned. Remember that values stored with **set** remain across merges whereas values stored with **identify** are stored for the duration of the current merge only. If the right hand side of the equality begins with a ª?º then an attention panel will be brought up to ask for the valueÐthe text following the ª?º is used as a prompt in the panel. This part of the **set** command is not yet fully implemented, so don't use it yet!

next

If a MiscMergeDriver initiated the merge, then the **next** command will cause the next merge dictionary to be loaded. This allows merges for multiple records to be placed into a single output merge. Note that the MiscMergeDriver will add in empty output strings as placeholders for the output List, since the merge for the next record will not be performed. This command might be used, for example, in creating pages of address labels. You could put several labels on one page this way.

omit

Causes the merge to be aborted. The output string will be empty. This command can be used in an **if** statement to conditionally abort merges depending upon values of keys in the merge dictionary.

ask

Prompts the user (via an alert panel) for the value of a field in the merge. **Warning:** This command is not fully implemented yet and requires the (not yet available) `_MiscMergeQuery.nib` file to be included in your project.