Introduction to RAPTOR: Data Files

Creating and Displaying Data Files

In RAPTOR we can create data files and read from the files. However, sorting, inserting records, or merging two data files requires some fancy footwork but if you're very ambitious, try it!

The `Redirect_Output` Procedure

To create a data file, you use a **`Call`** to the **`Redirect_Output`** procedure. RAPTOR provides two versions of this procedure.

1. A filename is used as an argument to **`Redirect_Output`**, as shown in the following examples:
   - **`Redirect_Output("sample.txt")`**
   - **`Redirect_Output("C:\MyDocuments\John.Doe\sample")`**

   Note that in the first example, only the filename is given. In this case, the specified text file will be created in the same directory as the current RAPTOR program. In the second example, the full path to the file is given. Also, in the second example, no file extension is specified. In this case, the file **`sample`** will be created with no extension.

2. You can either turn on or off **`Redirect_Output`** by including a simple **`yes/true`** or **`no/false`** argument, as follows:
   - **`Redirect_Output(True)`** or **`Redirect_Output(yes)`**
   - **`Redirect_Output(False)`** or **`Redirect_Output(no)`**

Now, the output must be redirected to the data file by using a **`Call`** to the **`Redirect_Output`** procedure. The name of the data file is used as the argument. This filename must be inside quotation marks (as shown in the examples above).

Next, create the code to input data. One variable is required for each field of the records in the data file. The **`Output`** box will **`PUT`** the value of those variables into each record. For example, to

create a data file with records that have two fields, **Name** and **Salary**, two variables are required (probably called **Name** and **Salary**). As values for different employees are input, each **Name** and **Salary** will be stored in the data file on a single line.

After the data has been entered, the **Redirect_Output** must be turned off. A **Call** is used to call the **Redirect_Output** procedure again, but this time it's turned off by using either **no** or **false** as the argument.

Figure 1 (following page) shows a RAPTOR flowchart that will write two records, each with two fields, to a data file named **sample.txt**. Figure 2 shows the contents of the file created (and opened in Notepad).

The Redirect_Input Procedure

To display the contents of a data file, the **Redirect_Input** procedure is used. This works similarly to the **Redirect_Output** procedure.

In a **Call** to **Redirect_Input**, the filename of the file to be read is used as the argument as follows:

```
Redirect_Input("sample.txt")
```

The records are read, normally, within a loop. This is accomplished with **GET** statements. **Input** boxes are used to **GET** each record (in this example, the records consist of the names and salaries). Nothing needs to be entered as a prompt. **Output** boxes are used to display the output of each record. The output is displayed in the **Master Console**.

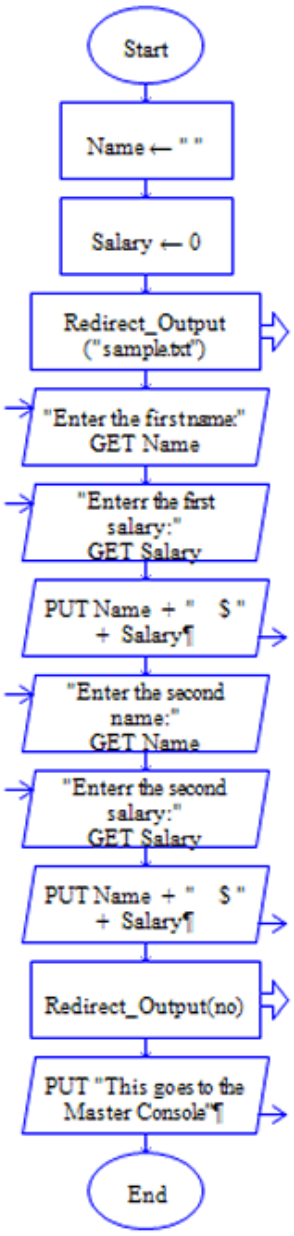

Figure 1 Program to write records to a data file

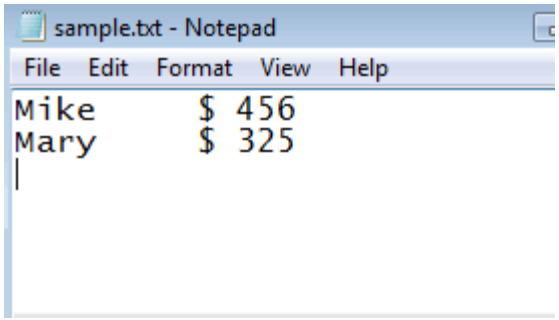

Figure 2 Text file created

The `End_Of_Input` Function

RAPTOR's built-in function, **`End_Of_Input`**, can be used as the test condition of a loop. When reading records in a data file, if this function is used as the test condition, RAPTOR will end the loop when all the records have been read.

When all the records have been read and written to the **`Master Console`**, the **`Redirect_Input`** procedure must be turned off with a **`Call`** to the procedure using **`False`** or **`no`** for the argument.

How the Contents of the File are Stored

The **`Redirect_Input`** procedure does not separate each field in a record. Rather, each record is stored as one line of string data. Each **`Input`** line reads all the fields in one record (or everything that is on a single line in the data file). Therefore, the records can be output to the **`Master Console`** but the fields cannot be manipulated easily to sort, insert, or merge. This can be done, but it requires advanced programming.

Figure 3 shows a sample of the code to read records from a data file (**`sample.txt`**) and the **`Master Console`** display.

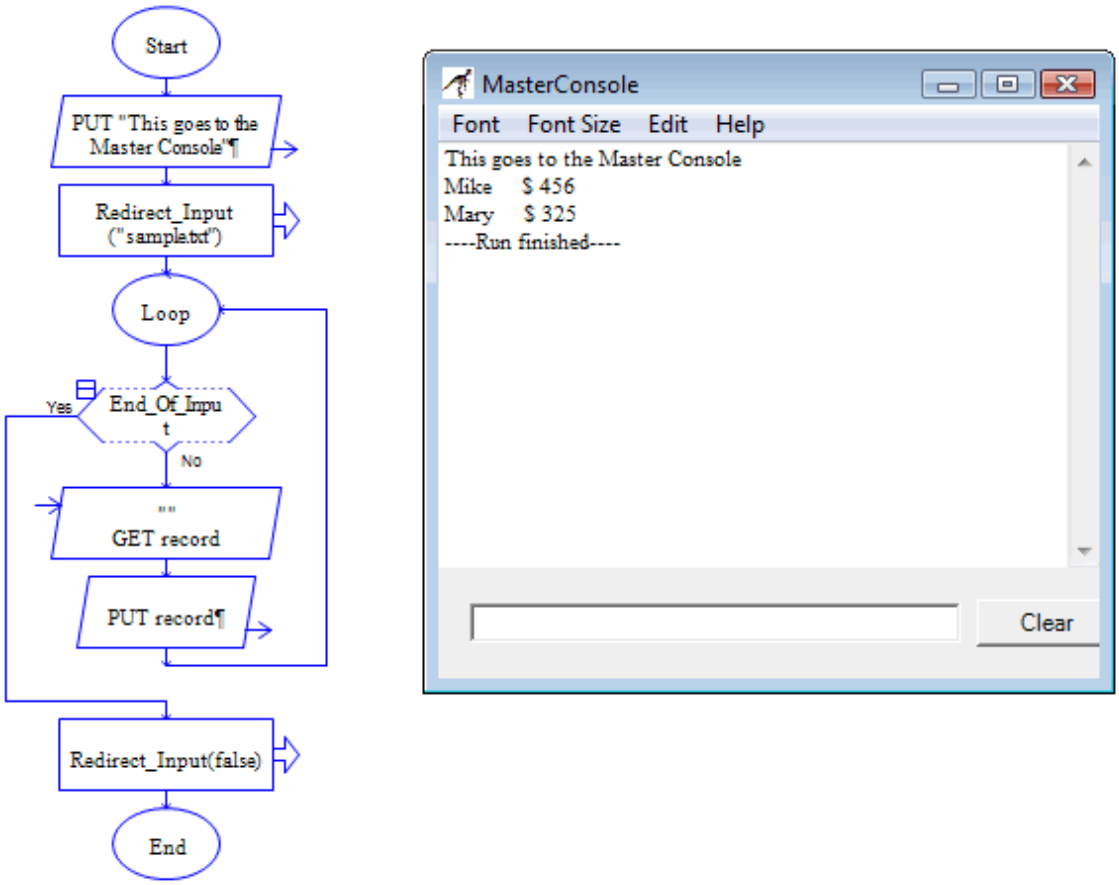

Figure 3 Reading records from a data file and displaying them