Journal of International Technology and Information Management

Volume 28 | Issue 1 Article 3

5-1-2019

The Use of Generic Scripting in Certain Application Development Projects

Vance Allen Etnyre
University of Houston-Clear Lake, etnyre@uhcl.edu

Jian Denny Lin
University of Houston-Clear Lake, linjian@uhcl.edu

Nanfei Sun
University of Houston Clear Lake, sun@uhcl.edu

The Use of Generic Scripting in Certain Application Development Projects

Vance Allen Etnyre
(University of Houston-Clear Lake)

Jian (Denny) Lin
(University of Houston-Clear Lake)

Nanfei Sun
(University of Houston Clear Lake)

ABSTRACT

This article discusses generic scripting, a useful scripting technology for developing applications. In its simplest use, generic scripting can be used as a documentation tool to replace flowcharting or pseudocode. in certain situations, generic scripting can lead directly to a working application without the need to write, compile and test a new program. Generic scripting is discussed as a tool which can be used in logical design, detailed design and implementation of a new application. An example is presented to show how generic scripting can be used by non-programmers to simplify the development of applications. This example is used to teach business processes at the authors' university.

Keywords: Generic Scripting, Application Design, Process Documentation, Data Analysis, Application Development

INTRODUCTION

Generic scripting is a tool which can be used in several phases of the application development process. In application development, best practices include at least three steps: Logical Design; Detailed Design and Implementation of the design. Sometime these steps are done in purely sequential order starting with logical design and continuing through detailed design and ending with implementation. Sometimes these steps are used in an iterative or cyclical fashion where implementation of one aspect of the system opens questions and options for the design or implementation of other parts of the system. Agile development methods follow the iterative development approach. A recent trend in system development

ISSN: 1941-6679-On-line Copy

is to provide very simple user interfaces which allow non-programming users to execute common tasks without having to write a program. Generic scripting follows the trend toward iterative development and the trend toward simplified use by non-programming end users.

In the logical design phase, generic scripting can be used to identify and document the logical steps to be followed to complete an application much like the development of a recipe can lead to the successful introduction of a new meal in a restaurant. Generic scripting can be used to identify and define components used in the process just as a list of ingredients would be included in a recipe. Steps in the development process would be listed to complete the general design of the process just as steps would be listed in a recipe. Details would be added to convert the general design to a detailed design for both the application and the recipe. When the design process is complete, the application can be assigned to teams to complete the creation and testing of the application (or the new restaurant offering).

The script interpreter/implementor module must be embedded in or attached to an execution platform. The designer of the interpreter module should specify which scripting mechanisms will be handled by that module. Features which should be supported include: commenting; condition testing; repetition of execution and interactions with data sources. Details of these features are discussed further in appendix B of this paper. There must be a feature for defining variables and procedures. Applications usually contain a mixture of common procedures and unique procedures. As blocks of script are developed to create and implement specific steps in a development process, these blocks can be stored and then reused in other situations.

For reusable scripts to be effective there must be mechanisms to allow the saving of scripts, insertion of script blocks and the execution of defined procedures. Tested and stored blocks can be inserted into other script blocks or can be executed on command to perform common functions. These reusable blocks reduce the time needed to create detailed designs for each logical design. The script example shown in appendix A of this paper shows how stored script blocks can be inserted into.

The following section discusses common uses and common features of traditional scripting languages. An example of a generic scripting application follows that discussion.

ISSN: 1941-6679-On-line Copy

OVERVIEW OF TRADITIONAL SCRIPTING LANGUAGES

In computer systems discussions, scripting refers to a command structure which allows programmers or users to write instructions which can be interpreted and executed to develop and implement applications. There are several popular scripting languages used in application development. JavaScript is very popular in the development of web-based applications. PL-SQL is a scripting language used to create and modify Oracle databases. Perl provides powerful text processing facilities to manipulate text files. Python, one of the most popular programming languages today, is widely used in Data Analysis, Web Designs and other general-purpose programming.

Scripting languages are designed for different tasks than are general programming languages or system programming languages, such as Java or C#. System programming languages are designed for building data structures and algorithms from scratch, starting from the most primitive computer elements such as words of memory. In contrast, scripting languages are designed for gluing: They assume the existence of a set of powerful components and are intended primarily for connecting components [Ousterhout, 1998]. Because of the existence of the prefabricated components, scripting languages provide means to build flexible applications quickly.

Scripting languages are often interpreted rather than compiled. A special program called an interpreter directly converts the script into execution steps on the computer, such that the execution of tasks could alternatively be executed one-by-one by a human operator.

Specific scripting languages have been used to implement specific tasks in a variety of systems [Beazley, 1996; Beazley and Lomdahl, 1997; Bakker and Jain, 2002; Arafa et al., 2003; Furr et al., 2009; Williams et al., 2010]. In [Casey et al., 2005], the researchers are focusing on super node implementations. Also, many of researches are conducted to reduce or eliminate dynamic type checks in compilers or virtual machines for dynamically typed languages [Ertl et al., 2002; Biggar et al., 2009; Gal et al., 2009; Rigo, 2004].

In the business domain, SAP script is the SAP System's own text-processing system. It is used to print pre-formatted text in pre-formatted forms.

WHEN IS GENERIC SCRIPTING USEFUL?

Generic scripting is most effective when it can be quickly transformed into actions. One way to do this is to have a programmer who is familiar with the scripting language use his or her programming skills to translate the script into programming statements. Then the programming statements can be compiled and tested. After successful testing, the program can be integrated into the existing system to produce the required output.

When a programmer translates a detailed design script into computer \source code", then another device (a compiler) must be used to translate (compile) the source code into executable code and the executable code must be added to the structure of executable code available for use within the system. To maintain system integrity, the new scope of executable code must be published (made available) to legitimate users of the system. If the new application will be reused within the system, the newly rebuilt system will require additional documentation and maintenance. If there is very little expectation that the new application will be reused within the system, then the efforts to rebuild the system and provide additional documentation and maintenance will not be justified.

As an alternative, scripting languages might be used in a system with an embedded interpreter which can execute scripts directly to produce the required output. Such a system would not require a new program to be written, compiled, tested and then implemented. This eliminates the need to rebuild or reconfigure the production system for each new ad hoc application.

It should be noted that interpreters for generic scripts are not optimized for performance. If performance is important, compiled code should be used. When an application will be used frequently, the overhead of creating compiled code and rebuilding the system is often justified by the additional efficiency of the compiled code.

Data analysis packages can be very useful in discovering anomalies or hidden patterns in data. Such packages usually require a certain amount of pre-processing of the data and a certain level of expertise in analyzing data. If such packages are not available or not configured for analyzing transactional data, a short script and an embedded interpreter can be very useful to transform data for analysis and display. The script statements in appendix B can be used for such a purpose.

ISSN: 1941-6679-On-line Copy

When an application is truly ad hoc with no expectation of repeating and it can be accomplished using a script plus the tools of a built-in interpreter, a non-programming manager can save a lot of time and save the overhead expense involved in changing the existing system to add a new program. This might be true for ad hoc analyses of sales data. An example of this situation is given below.

AD HOC ANALYSIS OF DATA - AN EXAMPLE

The following example is used to teach business process management in two graduate level courses. In this example, a sales manager of a small retail store is concerned that employees are abusing the policy of providing discounts to increase sales. A policy might allow discounts to sell high-end merchandise, but only up to a certain limit. The sales manager at a small retail bike shop has set a limit of 20% discounts to increase sales volume while maintaining an acceptable profit margin on sales. For a particular week, the manager feels that the overall profit margin for the store is unusually low and he suspects that sales persons may have been granting excessive discounts to bolster their sales volumes. He would like to know which sales persons were involved in improper discounts and which customers were involved in improper discounts.

The manager does not want to go through the entire set of sales data by hand to test his suspicions. As a non-programmer, he does not have the skills to create a new computer program to analyze all of the transactions for the week to test his suspicions. To start the process, the manager could use simple scripting to document the process used to prepare a display of sales data by employee and by customer.

After the manager wrote some simple scripts, he could ask a programmer to use the scripts as guides to write a program to extract and summarize data and to create simple charts. This would mean acquiring the services of a programmer, communicating the nature of the problem to the programmer, waiting for the programmer to write and test the code necessary to solve the problem and then waiting for the programmer to communicate the results.

As another possible option, the manager might be able to use an available interpreter to extract and summarize data and to create simple charts without needing the intervention of a programmer. Using either option, the results of the process might look like the next four charts (Figure 1-4).

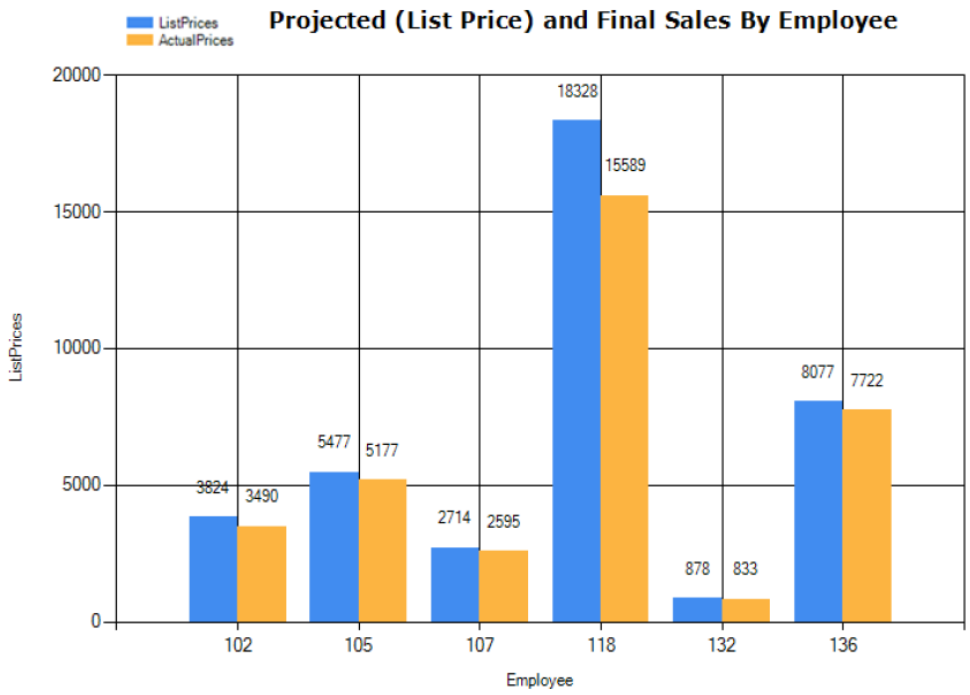
ISSN: 1941-6679-On-line Copy

The sales manager was interested in knowing if excess sales discounts were caused by any employee. The following script could be used to describe the process of obtaining the results sought by the manager.

Chart Title = Projected (List Price) and Final Sales By Employee
Chart Yvariables = 2 Chartype = Bar
Data select Customer as x1, ListPrices as y1 from ETSL Data where docType= 213 order by employee
Data select Customer as x2, ActualPrices as y2 from ETSL Data where docType= 213 order by employee

The results sought by the manager are shown in figure 1.

Figure 1: Projected and Final Sales by Employee



It can be seen in figure 1 that employee 118 had very high sales during the week in question. The discount percentage for this employee is very close to the 20% maximum but this chart, by itself, is not conclusive.

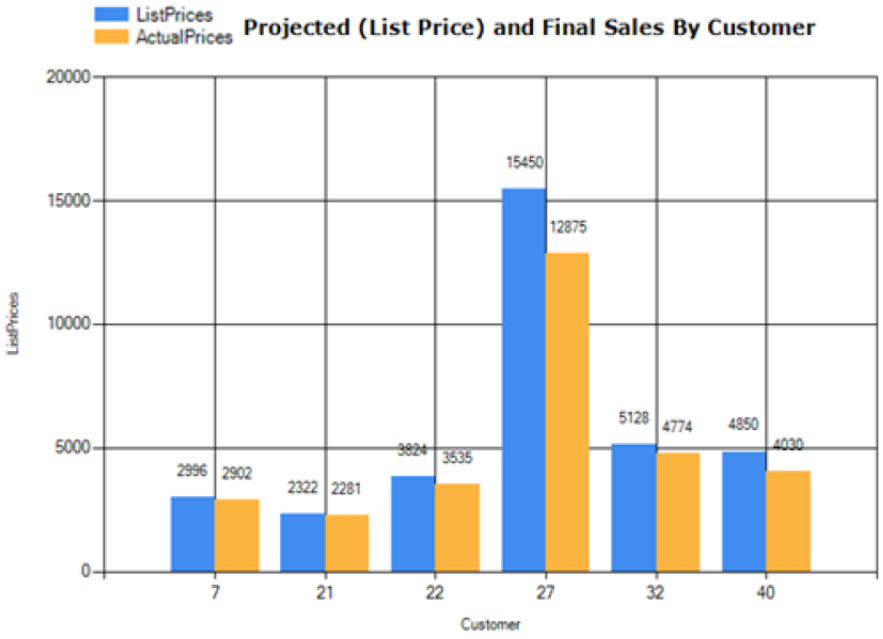
ISSN: 1941-6679-On-line Copy

The sales manager was interested in knowing if excess sales discounts were caused by any particular customer. The following script could be used to describe the process of obtaining and displaying the results sought by the manager.

```
Chart Title = Projected (List Price) and Final Sales By Customer
Chart Yvariables = 2 Chartype = Bar
Data select Customer as x1, ListPrices as y1 from ETSL Data where
docType= 212 order by customer
Data select Customer as x2, ActualPrices as y2 from ETSL Data where
docType= 212 order by customer
```

The results sought by the manager are shown in figure 2. It can be seen in figure 2 that Customer 27 made very high purchases during the week in question. The discount percentage for this customer was very close to the 20% maximum, but again, by themselves, these results are not conclusive.

Figure 2. Projected and Final Sales by Customer



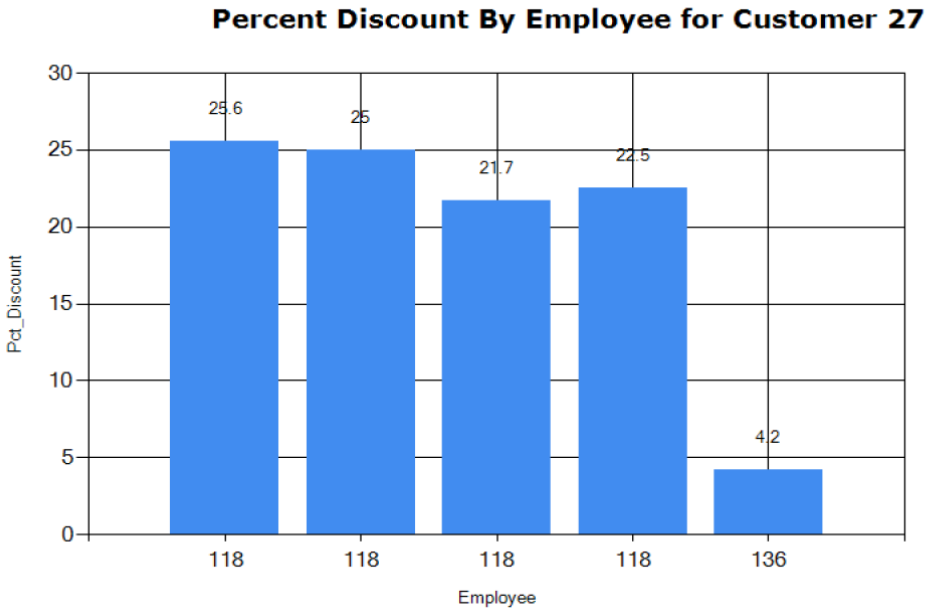
After looking at the two factors individually, the sales manager was interested in knowing if there was a connection between the sales to customer 27 and sales made by employee 118. This could be accomplished by taking a careful look at

the sales made to customer 27 during the week in question. This was done with a simple request to analyze the sales records for customer 27. The script for this request was:

```
Chart Title = Percent Discount by Employee for Customer 27
Chart Yvariables = 1 Chartype = Bar
select Employee as x1, Pct_Discount as y1 from Sales Data where
docType =21 and Customer = 27
```

The results of this inquiry are shown in Figure 3.

Figure 3. A Figure of Percent Discount by Employee for Customer



It can be seen in figure 3 that customer 27 had 5 sales within the selected week. One sale was a sale made by employee 136 with a very small discount. The remaining four sales were made by employee 118. Each of these sales had discounts which exceeded the 20% limit.

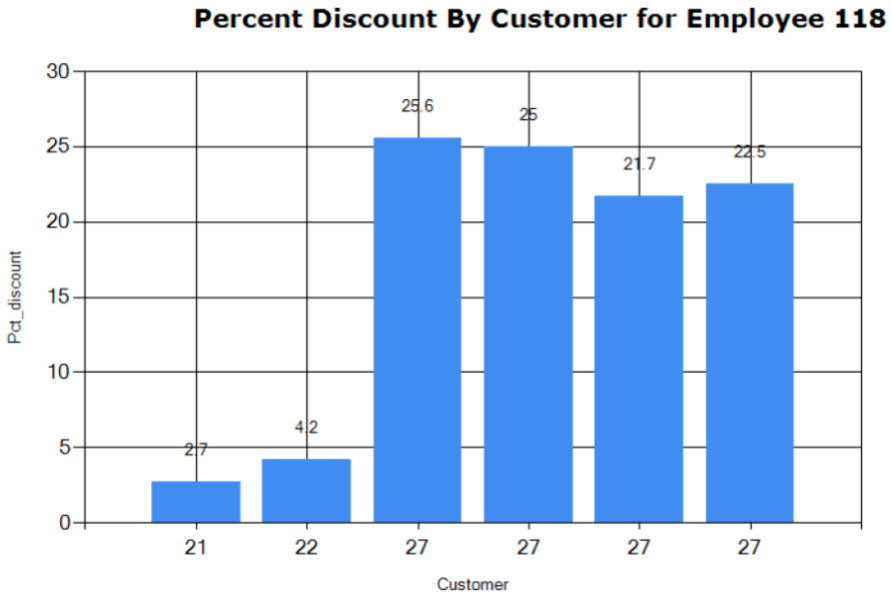
The last factor to be evaluated by the manager was a detailed analysis of sales by employee 118. The request to look at the sales made by employee 118 was accomplished with the simple script:

ISSN: 1941-6679-On-line Copy

Chart Title = Percent Discount By Customer for Employee 118
Chart Yvariables = 1 Chartype = Bar
select Customer as x1, Pct_Discount as y1 from ETSL Data where docType =21 and employee = 118

The results of this inquiry are shown in Figure 4.

Figure 4. A Figure of Percent Discount by Customer for Employee



It can be seen in figure 4 that employee 118 had 6 sales within the selected week. The first two were sales with very small percentage discounts. The remaining four sales were to customer 27. Each of these sales had a discount which exceeded the 20% limit. The largest discount was 25.6%. Since the analysis was done within an embedded program in an existing system, the manager could click on this bar in the bar chart and drill down to the details of this sales transaction. The results of this drill down process can be presented in the formatted display shown in figure 5.

ISSN: 1941-6679-On-line Copy

Figure 5. Sales Order Details

Figure 5 shows a sales order for a high-end off-road bicycle, sold at a substantially discounted price plus a $50.00 helmet which was thrown in for free. The total discount of $500.00 was 25.6% of $3,450, the total of list prices.

In this example, the sales manager was able to indicate his desire to chart sales discounts by employee and by customer using simple scripting statements. If the system used by the manager contained an embedded interpreter to process the script statements directly, the manager could have gotten his information directly from the system without requiring a programmer to create and test separate programs. The program used in this example is named LocalBikes. It was written by graduate students at the University of Houston - Clear Lake (UHCL). It is used in two courses within the College of Business at UHCL to teach business processes. It was specifically designed as a small retail complement to the Global Bikes program provided by SAP to teach business processes. Students in the College of Business who are non-programmers use the LocalBikes program and its embedded scripting capability to see the results of business processes quickly and easily compared to using SAP's Global Bikes program. As shown in the script segments above, using generic scripting with an embedded interpreter can provide a very simple option for

ISSN: 1941-6679-On-line Copy

developing applications in certain situations. This option does not require writing a new program, compiling and testing the new program and rebuilding the host system to include the new program.

One way to demonstrate the usefulness of generic scripting is to use a generic \meta script". In situations where an interpreter does not exist for application scripts, the services of a programmer must be used. A generic \meta script" for this process of designing and implementing a new application might be written as:

```
Manager produce logical design for application
Manager give logical design for application to developer
Developer convert logical design into detailed design
Developer give logical design for application to programmer
Programmer write new program to implement detailed design
Programmer compile and test new program
Programmer rebuild system to include new program
Manager use new program to implement application
```

In situations where an interpreter exists for application scripts, a generic meta script for the process of designing and implementing a new application might be written as:

```
Manager produce logical design for application
Manager convert logical design into detailed design generic script
If application can be executed directly from a script
        Manager or programmer modify the generic script to _t the
        interpreter
        Manager or programmer run the modified script to implement the
        application

Else //do it the old way if interpreter not available
        produce logical design for application
        Manager give logical design for application to developer
        Developer convert logical design into detailed design
         give logical design for application to programmer
        Programmer write new program to implement detailed design
        Programmer compile and test new program
        Programmer rebuild system to include new program
        Manager use new program to implement application
```

REFERENCES

Ousterhout, J. K. (1998). Scripting: Higher Level Programming for the 21st Century. IEEE Journal Computer, Vol. 31, Issue 3, pages 23-30.

Beazley, D. M. (1996). SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++. Proceedings of the USENIX Fourth Annual Tcl/Tk Workshop.

Beazley, D.M. and Lomdahl, P.S. (1997). Building Flexible Large-Scale Scientific Computing Applications with Scripting Languages. Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing.

Bakker, J L. and Jain, R. (2002). Next generation service creation using XML scripting languages. Proceedings of The IEEE International Conference on Communications.

Arafa, Y. and Kamyab, K. and Mamdani, E. (2003). Character animation scripting languages: a comparison. Proceedings of the second international joint conference on Autonomous agents and multiagent systems

Furr, M., An, J. D. and Foster, J. S. (2009). Profile-guided static typing for dynamic scripting languages. Proceedings of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications.

Williams, K., McCandless, J. and Gregg, D. (2010). Dynamic interpretation for dynamic scripting languages. Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization.

Ertl, M. A., Gregg, D., Krall, A. and Paysan, B. V. (2002). A generator of efficient virtual machine interpreters. Journal of Software: Practice and Experience., Vol. 32, Issue 3, pages 265-294.

Casey, K., Gregg, D. and Ertl, M. A. (2005). Tiger - an interpreter generation tool. Proceedings of the 14th international conference on Compiler Construction.

Biggar, P., deVries, E., and Gregg, D. (2009). A practical solution for scripting language compilers. Proceedings of the 2009 ACM symposium on Applied computing.

  ISSN: 1941-6679-On-line Copy

Gal, A., Eich, B., Shaver, M., Anderson, D., Mandelin, D., Haghighat, M. R., Kaplan, B., Hoare, G., Zbarsky, B., Orendorff, J., Ruderman, J., Smith, E. W., Reitmaier, R., Bebenita, M., Change, M., and Franz, M. (2009). Trace-based just-in-time type specialization for dynamic languages. Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation.

Rigo, A. (2004) Representation-based just-in-time specialization and the psyco prototype for python. Proceedings of the 2004 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation.

APPENDIX A.

Definition of the pretested block of script ETSL Summarize

```
1       //ETSL Summarize
2       Begin Summarize
3          select * from inputFile order by Group
4          copy data1 row 1 to targetrow
5          ForEach row in data1
6             copy currentrow to testrow
7             If sameGroup (Group in testrow = Group in targetrow)
8                Begin sameGroup.true
9                   sum data1 column 8 into targetrow column 8
10                  sum data1 column 9 into targetrow column 9
11               End sameGroup.true
12            Else
13               Begin sameGroup.false
14                  set column 0 in targetrow = Group in targetrow
15                  set column 1 in targetrow = dataType
16                  clean targetRow
17                  copy targetrow into data4
18                  copy testrow to targetrow
19               End sameGroup.false
20            End sameGroup
21            next row in data 1
22         End ForEach //row in data 1
23      End Summarize
```

This pre-tested script module defines the basic logic of the summarize transform. It works within the script interpreter embedded in the LocalBikes program. This script segment requires the following data components: inputFile, outputFile, data1, data4, dataType, currentRow, targetRow, testRow and Group which are defined within the LocalBikes program.

The logic shown above can be customized by inserting it into a script block which sets the values of the required parameters (inputFile and Group). The script file below customizes the previous script block to summarize a file called SalesData by Customer.

```
Transform Sales Details as data1 into data4 as ETSL Data with datatype
212
Group by Customer data1 column 2
Summarize columns 8 and 9
Load Script ETSL Summarize from ETSL Scripts //insert all lines here
Save daa4 as outputFile
End Transform
```

APPENDIX B.

Components of a Generic Scripting Language

Features which should be supported in a generic scripting language include: commenting; condition testing; repetition of execution and interactions with data sources. For reusable scripts to be effective there must be mechanisms to allow the saving of scripts, insertion of script blocks and the execution of defined procedures. It should be noted that line numbers are not required but are added to this example to facilitate explanation of the script syntax and components. In this example, line numbers refer to the line numbers shown in the script block ETSL Summarize. These statements will be interpreted and executed within a module of the LocalBikes program described in Appendix C.

Statement syntax: Statements begin with command words or declarative words and continue for the remainder of the line. Words such as select (line 3), copy (line 4) or set (lines 14 and 15) are command words. Words such as Begin (lines 2, 8 and 13), End (lines 19, 20 and 22), ForEach (line 5) and If (line 7) define the structure and connectivity of the syntax.

ISSN: 1941-6679-On-line Copy

Commenting: Non-executable comments should be encouraged to promote common understanding of the script's intended function. This example uses // to start a comment as shown in lines 1 and 22.
Condition testing: Conditions must be tested (as in line 7) and alternate paths should be followed depending on the state of the condition (as in lines 8-11 and 13-19).

Repetition of execution: Blocks of statements may be executed repeatedly depending on certain conditions. Lines 5-22 show repetition of statements until all rows have been processed.

Interaction with data sources: Line 3 defines a data selection to be implemented by the interpreter/implementor program or module (in this example, a module within the LocalBikes program).

APPENDIX C.

The Local Bikes program

The LocalBikes program is a business processing program which was designed, programmed and implemented by graduate students at University of Houston - Clear Lake. LocalBikes was designed to complement the SAP GlobalBikes program used to support teaching of business processes. The LocalBikes program was written in C# and compiled on the Microsoft Visual Studio platform by students in the MS-MIS program at University of Houston - Clear Lake. The program has been modified as a learning platform by many student teams.

The scripting portion of the LocalBikes program features an interpreter which reads each line of script and acts on the "command word", the first word of each line. The lines of script are routed to various modules by a switch command which acts on the first word of the script line. The quality of the code is very inconsistent, but the program has served well as an instructional tool. Generic scripting has been part of the system design courses for the last six semesters at University of Houston - Clear Lake.

ISSN: 1941-6679-On-line Copy