Muddle and Puddle

Muddle is a utility which is used to generate the data files for the Habitat C64 object disk. It reads a description file written in the Muddle language and from this generates a series of data files plus a textual listing showing what it has done.

The ’muddle’ command itself has the following from:

```
        muddle < mudfile > listfile
```

I.e., it reads the Muddle input file from the standard input and writes the listing file to the standard output. It also generates five data files:

```
        class.dat       - object class descriptors
        image.dat       - object images
        sound.dat       - sound effects
        action.dat      - object behaviors
        head.dat        - head images
```

Collectively, these five ".dat" files define the C64 object database. Muddle also generates three other output files:

```
        class_equates.m - a Macross include file defining symbolic constants
                                for the various class names
        width.incl.pl1  - a PL/1 include file defining width tables for the
                                host’s collision detection routines
        capacity.incl.pl1 - a PL/1 include file defining size tables for
                                the host’s memory capacity monitor routines
```

The Muddle Language

The input to Muddle is a Muddle file in the Muddle language. This is a very simple language that allows you to lay out class definitions quickly and easily. A Muddle file consists of a series of class definitions, each of which specifies the elements (image files, sound effects, data bytes, and so on) that go into a particular Habitat object class. The elements themselves are either specified literally (in the case of byte and word data) or taken from data files whose names you give (in the case of images, sounds and the like). Since certain class elements, notably some behavior code, are shared among multiple classes, Muddle allows you to define the class elements separately from the classes themselves, and then refer to the elements symbolically in more than one class definition statement.

There are thus two types of statements in the Muddle language, the "class definition statement" and the "element definition statement".

The element definition statement has the form:

```
        <elementType> {
                <elementSpecifier>
                <elementSpecifier>
                ...
        }
```

where <elementType> is one of the element types ’image’, ’action’, ’sound’ or ’head’; an <elementSpecifier> has the form:

```
        <label> : <source>
```

or

```
        <label> : <source> <width>
```

or

```
        <label> : <source> <width> <standoffset>
```

where <label> is a symbolic name to be given to the element; <source> is the source of the element, either the symbolic name of another element or the name of a Unix file, enclosed in quotes; and <width> and <standoffset> are numbers that may be given, for ’image’ elements only, that describe the graphic characteristics of the image.

The class definition statement has the form:

```
        class <className> <classNumber> {
                <classElement>
                <classElement>
                ...
        }
```

where <className> is a symbolic identifier to be used as the clas name; <classNumber> is the class number of the class; a <classElement> has the form:

```
        <elementType> <source>
```

or

```
        <elementType> <dataList>
```

where <elementType> is ’image’, ’action’, ’sound’, ’head’, ’byte’ or ’word’; <source> is the source of the element, either a Unix file name enclosed in quotes or the symbolic name of an element defined by an element definition statement; and <dataList> is simply a comma-separated list of numbers that are used directly for byte and word data (this data is placed directly into the class descriptor for the class).

Output

Muddles writes a listing to the standard output and produces the five data files named "class.dat", "action.dat", "image.dat" and "sound.dat" and "head.dat". Each data file consists of a 256 word table followed by the data record. Each data record begins with a length word, followed by the actual data itself. The data records in the "class.dat" file are class descriptors. They have this structure:

```
        word    class descriptor length (i.e., the length word)
        byte    number of images
        byte    number of sounds
        byte    number of actions
        byte    offset from start of class descriptor to first image
        byte    offset from start of class descriptor to first sound
        byte    offset from start of class descriptor to first action
        word    instance descriptor length
        bytes   random class-specific data (’word’ and ’byte’ class elements)
        bytes   image numbers
        bytes   sound numbers
        bytes   action numbers
```

Puddle

Puddle is a Muddle post-processor that is used to cope with changes that are made to the object disk database over time. Puddle is invoked with the command

```
        puddle old.dat new.dat > composite.dat
```

where "old.dat" is an old data file, "new.dat" is a newer version of the same file, and "composite.dat" is the output that Puddle will produce which is a new data file with the contents of "new.dat" but with a layout that minimizes the number of blocks that are different from "old.dat". (In order to use this effectively, your Muddle input file must be maintained in such a fashion that new elements are always added at the end, so that the resource ID numbers of corresponding elements in "old.dat" and "new.dat" are the same.)

Once the composite output file is created with Puddle, you can use the utility ’deltab’ (short for delta-block) to generate a listing of the block numbers that have changed:

```
        deltab old.dat composite.dat > listfile
```

where "old.dat" and "composite.dat" are the same files used in the ’puddle’ command. ’listfile’ is simply a list of the block number (in C64 disk blocks) that differ between the two ".dat" files.