

Platform to automate creation of serialised data objects for import into a game engine

FIELD OF INVENTION

[001] The present disclosure relates to a platform to receive external data stored in tables and
5 generates code to extract the external data for import into a game engine environment.

BACKGROUND

[002] While software game engines facilitate game development through access to game
development tools, a developer still has the tedious job of customising these tools in accordance with
10 game requirements. Part of this customisation is providing these tools with data parameters that are
used by the game being developed.

[003] A direct approach is to customise these development tools within the game development
environment, where a developer manually changes all data values individually within the software game
engine itself. This direct approach is tedious and time consuming.

15 [004] A second approach is through batch modification through importing tables containing the data
parameters for use by the game development tools. One or more spreadsheets (e.g. in Excel® or Google
Sheets® format) are used to edit and store external data in tabular format. To read the external data in
the game engine environment, the following procedure is generally followed: (1) write code regarding
the specific content of the data to read the external data for conversion into in-game data objects; (2)
20 use tools to convert the external data into easy-to-read data format (such as CSV), and then write code
to read the data objects into the game engine environment. Programmers may need to re-write code for
each new table.

[005] The above process requires coordination between designers and programmers, which leads to
time cost and inefficiency in maintenance of the data structure used in the spreadsheets containing the
25 external data. Once the data structure changes, programmers need to adapt their already written code to
accordingly, otherwise there will be an error when of the created data objects are read in game.

[006] An object of the present invention is to provide a solution that addresses the above shortcomings.

SUMMARY OF THE INVENTION

30 [007] According to a first aspect of the present invention, there is provided a platform for creating
serialised data objects from external data, the platform configured to generate code that analyses a table,
comprising the external data organised in cells, to identify the type of data stored in each cell;
determines if the identified data types are parsable; generates, in response to the identified data types
being parsable, further code that when compiled outputs at least one serialised data object, wherein the
35 data stored in each cell of the table is assigned to an attribute of the serialised data object; and retrieves
an export format for the serialised data object, wherein the platform is further configured to compile the
generated further code to output the serialised data object in the retrieved export format.

[008] According to a second aspect of the present invention, there is provided a method for creating serialised data objects from external data, the method comprising the steps of: generating code to analyse a table, comprising the external data organised in cells, to identify the type of data stored in each cell to determine if the identified data types are parsable; generating, in response to the identified data types being parsable, further code that when compiled outputs at least one serialised data object, wherein the data stored in each cell of the table is assigned to an attribute of the serialised data object; retrieving an export format for the serialised data object; and compiling the generated further code to output the serialised data object in the retrieved export format.

10 BRIEF DESCRIPTION OF THE DRAWINGS

[009] Representative embodiments of the present invention are herein described, by way of example only, with reference to the accompanying drawings, wherein:

[010] Figure 1 shows a workflow to import external data into a software game engine.

15 [011] Figure 2A shows a spreadsheet that is parsed during the workflow of Figure 1.

[012] Figure 2B shows another spreadsheet that is also parsable using the workflow of Figure 1.

[013] Figure 3 shows code generated during the workflow of Figure 1 that is used to parse the spreadsheet of Figure 2A.

20 [014] Figure 4 shows code generated during the workflow of Figure 1 that is used to output serialised data objects.

[015] Figure 5 shows a serialised data object in scriptable object format output by the generated code of Figure 4.

[016] Figure 6 shows a serialised data object in JSON format output by the generated code of Figure 4.

25 [017] Figure 7 shows a serialised data object in CSV format output by the generated code of Figure 4.

[018] Figure 8 shows generated code by the platform that manages retrieval of spreadsheets used to contain the external data.

30 DETAILED DESCRIPTION

[019] In the following description, various embodiments are described with reference to the drawings, where like reference characters generally refer to the same parts throughout the different views.

35 [020] The present application falls within the field of game development using a game engine. A game engine provides a suite of reusable software components that a game developer needs to build a game, such as graphics, sound, physics and artificial-intelligence (AI) functions. This software suite allows the game developer to create game objects, which is a representation of an element designed for

a game, such as characters, weapons and ambient objects. Changing the attributes for the game objects allows for each element to be customised to meet the needs of a game.

[021] Data for these in game objects, used to modify their attributes, may be stored in external files, such as spreadsheets created by sheet software like Excel® and Google Sheets®. Spreadsheets are used because they allow the external data to be organised in tables. However, such spreadsheets cannot be imported directly into the game engine environment. Specific code has to be written to parse the table to create data objects in a format that is recognised by the game engine, with each of the data objects containing the relevant external data intended for a game object. The data objects can then be imported into the game engine, meeting the objective of imported the external data into the game engine for conversion into the game object.

[022] Rather than having to manually rewrite code to parse each table used to store external data, the present application seeks to implement a workflow that automates the procedure of importing data stored in external tables. The present application achieves this through a platform hosting a program that upon detecting that external data for import into a game engine is contained in a table, creates code for the table that is recognised by the game engine.

[023] In more detail, the program generates code that can access the external data that is stored in the table in cells that are organised into rows and columns. The generated code is configured to identify the type of data stored in each cell and outputs executable statements catered for that type of data. For example, if the cell is identified to contain an integer, the generated code will output executable statements to cater for data of integer type. In one implementation, the generated code identifies or learns the type of data stored in each of the cells of the table from an index that accompanies the table, the index defining the data stored in each of the cells in the table through stating one or more properties of the stored data. For example, the index may define that the data stored in a particular cell should be an integer and is intended for use at only a client.

[024] Upon determining that the identified data types present across the external data is parsable, the platform generates further code to output the executable statements catered for each of the identified data types to serialise the external data stored in the cells of the table. This serialisation creates one or more serialised data objects, which are data objects each having a unique identifier and used to combine different data types (e.g. string, float, array). Each data type is identified by its name and the different data types are also called the attributes for each data object. The generated further code outputs executable statements that map the data in each cell of the table to an attribute of the serialised data object, so that when the generated further code is compiled, the data stored in each cell of the table is assigned to an attribute of the serialised data object, so that the serialised data object has sets of name and data pairs. The name for each attribute of the serialised data object is obtained from a field name for the cell providing the data for that attribute. The generated code also exports the one or more serialised data objects in a stipulated export format to accommodate various acceptable input formats that different game engines have. A game engine is then able to load the one or more serialised data

objects for use with matching in game data objects, whereby the data in the attributes of each serialised data object is copied to a corresponding attribute of the game object, for which the serialised data object is intended. In this manner, attributes of programmed in game data objects may be modified by external data in spreadsheets.

5 [025] Advantageously, all code generation and the output of one or more serialised data objects is automatic, without requiring code to be written by the game developer, thereby reducing the workload to manually create code that can export external data stored in spreadsheet documents during the game development stage

[026] The automatic code generator to output serialised data objects is described in greater detail
10 below in conjunction with Figures 1 to 8.

[027] Figure 1 shows a workflow 100 to import external data into a software game engine, such as but not limited to Unity®. Steps 102 and 104 are directed at the design of a spreadsheet that is used to contain the external data. Steps 106, 108, 110, 112 and 114 are performed by a platform hosting an automatic code generator, in accordance with an embodiment of the invention, to create serialised data
15 objects from the external data contained in the spreadsheet from the steps 102 and 104. Steps 116 and 118 are performed by the software game engine, which imports the serialised data objects created in the step 114.

[028] In the steps 102 and 104, a methodology is adopted where the data structure of a spreadsheet is defined and the external data is arranged to follow the data structure. This methodology, which for
20 example predefines how the external data is to be organised into columns of specific data type, standardises the creation of spreadsheets that will allow the generated code of the platform to work with the spreadsheet.

[029] Figure 2A shows a spreadsheet 200 (shown as two halves due to space constraints) configured in accordance with one possible implementation of this methodology. The spreadsheet 200 has a table
25 202 comprising the external data organised in cells 206. The spreadsheet 200 also contains an index 204 accompanying the table 202.

[030] The index 204 defines the data stored in each of the cells 206 in the table 202 through stating one or more properties 208 of the stored data. The index 204 is referred to determine which of the one or more properties 208 impact the code generation that occurs in the steps 106, 108, 110, 112 to create
30 serialised data objects. The one or more properties specify one or more of, but not limited to: a field name for each of the cells 206, the type of data stored in the cell 206, and whether the data in the cell 206 is for use on a client or a server. For instance, all cells under the column “Example_Float” has four properties: a specification (“ID”) of a unique identifier of the serialised data object to which the data in the cell belongs; a specification (“Example_Float”) of a name for the data stored in the cell which also
35 provides the name (see 508 in Figure 5, 608 in Figure 6 and 708 in Figure 7) of an attribute of the serialised data object to which the data in the cell is assigned; a specification (“float”) of the type of data stored in the cell, being in this case float data type; and a specification (“both”) of the scope of

the data stored in the cell, which in this case indicates it is for use in a server or a client. Similarly, all cells under the column “Example_Vector2” has five properties: a specification (“ID”) of a unique identifier of the serialised data object to which the data in the cell belongs; a specification (“Example_Vector2”) of a name for the data stored in the cell which also provides the name (see 510 in Figure 5, 610 in Figure 6) of an attribute of the serialised data object to which the data in the cell is assigned; a specification (“Vector2”) of the type of data stored in the cell, being in this a two dimensional array; a specification (“client”) of the scope of the data stored in the cell, which in this case indicates it is for use in only a client; and a specification (“IsNullOrEmpty”) of whether a nullity check is to be performed to determine whether there is a requirement for the cells to store data, which in this case is performed for the cells that fall under the column “Example_Vector2”.

[031] The external data in the table 202 is considered validly arranged when it is organised in accordance with the structure defined by the index 204. For the spreadsheet 200 of Figure 2A, the structure is as follows. The first row specifies a field name for the data in the cells under its column. The second row specifies the type of data stored in the cells under its column. The third row specifies the scope of the data stored in the cells under its column, because some field names are only used by the client, while others are only used by the server for memory optimization. The fourth row provides a means to specify a test condition that can be imposed to check whether the data stored in the cells under its column is validly configured. The test condition illustrated in spreadsheet 200 is a nullity check (“IsNullOrEmpty”). However, other test conditions (not shown) may be specified, such as, but not limited to: whether there is a data type match in the cell 206 against the data type that the cell 206 is defined to store (e.g. if the second row specifies that the data type as being an integer, verifying that the data in the cell 206 is indeed of integer type and not a float type or string type) and whether there is conformity between the data in one or one more of the cells 206 that are interrelated, applicable when these one or more interrelated cells are required to define an attribute of a game object (e.g. if the data in one cell 206 indicates that the gender of a player character is a female, then test for whether the data in one or more interrelated cells 206 also contain data that supports the indicated gender, such as a female name).

[032] It will be appreciated that other structures are possible, depending on the content of the index 204. Figure 2B shows another spreadsheet 250 (shown as two halves due to space constraints). Like the spreadsheet 250 of Figure 2A, the spreadsheet 250 has a table 252 comprising the external data organised in cells 256. The spreadsheet 250 also contains an index 254 accompanying the table 252. The index 254 defines the data stored in each of the cells 256 in the table 252 through stating one or more properties 258 of the stored data. The index 254 thus defines the data structure of the spreadsheet 250, whereby the external data is arranged to follow the data structure during configuration of the spreadsheet 250.

[033] The index 254 of the spreadsheet 250 shares similar definitions with the index 250 of the spreadsheet 250, such as defining the presence of script specific array (key enum Color), integer, string, Boolean and float data types. However, the index 254 of the spreadsheet 250 also defines an array with multiple data sets (see 260A and 260B for the entry uniquely identified as “2”). When parsing the spreadsheet 250 in step 106, the code generated by the platform recognises, through reference to the index 254, that each cell under the column “Example_Class_Array” is an array having multiple data sets 260A and 260B and extracts all of them. In contrast, existing approaches only extract the first data set, but not both. The platform then flags each extracted data set 260A and 260B for assignment as an attribute to a serialised data object that is eventually output. In this manner, each cell under the column “Example_Class_Array” can produce more than one attribute for the serialised data object.

[034] Returning to Figure 1, the platform that performs steps 106, 108, 110, 112 and 114 may be a single computer server or several computer servers with at least one memory to store instructions to effect the generation of code that creates serialised data objects from the external data contained in a spreadsheet; and one or more processors that execute the instructions.

[035] In step 106, the platform generates code 300, shown in Figure 3, that is used to parse the spreadsheet 200. Segment 302 is a C# indexer, which serves to identify the cells 206 using their respective field name. Segment 304 is to create a map in runtime, to accelerate the identification of the cells 206. The generated code 300 analyses the table 202 to identify the type of data stored in each of the cells 206 to determine if the identified data types are parsable. The different data types that are present in the table 202 are identified through reference to the index 204 accompanying the table 202. The generated code 300 is configured to differentiate between the index 204 providing the data definitions and the table 202 comprising the external data. For example, there may be executable statements that set out the location of the index 204 and the location of the table 202 in the spreadsheet 200. As mentioned earlier, the index 204 defines the data stored in each of the cells 206 in the table 202 through stating one or more properties 208 of the stored data. Reference is made to the second row of the index 204 because it specifies the different data types stored in the cells 206.

[036] In the approach shown in Figure 2A, the generated code 300 supports basic data types, such as int, string, float, and bool. Customised data types commonly used in software game engines are also supported, such as “Vector2” (a two dimensional array), “Vector3” (a three dimensional array), and script specific arrays (e.g. “enum Gender”, “class User” and “repeated int”). The data type for the data stored in each of the cells is also reflected in the field name chosen for each of the data cells, where the field name is called during serialised data object creation (discussed later in step 106) or by an in game data object during external data import.

[037] In addition to determining whether the identified data types in the table 202 are parsable, further analyses performed on the table 202 content, in conjunction with the index 204, are as follow. The data in each cell 206 may be analysed for validity against its respective one or more properties 208 found in the index 204. For example, if one of the properties 208 specifies that the data in the cell 206 is an

array data type, then the cells 206 that store a different data type (e.g. an integer) have invalid data. A nullity check may be performed for cells 206 required to store data for the detection of blankness (i.e. whether the cell 206 indeed stores data). There may also be a check for duplicate field names. An error message may be output in response to any of these analyses returning results of non-compliance, such as the detection of blank cells 206 during the nullity check or the identified data types being determined to be imparsable. In one implementation, the returning of results of non-compliance of any one or more of these analyses may prevent the spreadsheet 200 from being parsed in step 106. Alternatively, these analyses may be done at a later part of the workflow 100, such as in step 118.

[038] If the spreadsheet 200 is parsable in step 106, the platform generates, in step 108, further code 400 shown in Figure 4. Compilation of the further code 400 in step 110 brings about data serialisation in step 112. These steps 108, 110 and 112 are described in greater detail as follows. The further code 400 combines the different data types 402 into a serialised data object, providing the serialised data object its attributes, and providing a unique identifier 404 for the serialised data object. The further code 400 combines the different data types 402 by calling their respective field names 406, 408, 410, 412, 414, 416, 418 and 420. It should be noted that while “string”, “int”, “bool”, “float”, “Vector2”, “Vector3”, “enum Gender”, “class User” and “repeated int” are the different data types shown in Figure 2A, the further code 400 of Figure 4 is for the creation of a serialised data object having a subset of these different data types, namely “string”, “int”, “float”, “Vector2”, “Vector3”, “enum Gender”, “class User” and “repeated int”. During the generation of the further code 400, the index 204 is analysed to determine a combination of different data types for use as the attributes for the serialised data object.

[039] This further code 400 when compiled then outputs at least one or more of such serialised data objects (see 522 and 524 in Figure 5; see 622 and 624 in Figure 6; and 722 and 724 in Figure 7), wherein the data stored in each cell 206 of the table 202 is assigned to an attribute (see 508, 510, 512, 514, 515 and 518 in Figure 5; 608, 610, 612, 614, 616 and 618 in Figure 6; and 708, 714, 716 and 718 in Figure 7) of the serialised data object (522, 524; 622, 624; and 722, 724). The name for each of the attributes for the serialised data object is obtained from a field name for the cell 206 providing the data for that attribute.

[040] The compilation of the further code 400 results in the output, in step 114, of serialised data objects in an export format, such as but not limited to JSON, scriptable object or CSV. The export format is retrieved, for example from a configuration file. In one implementation, the configuration file may be placed in the same folder where the spreadsheet 200 is stored.

[041] As an example, the configuration file has the following lines:

```
SERVER_LANGUAGE =GO
CLIENT_LANGUAGE =C#
FORMAT = ScriptableObject
SERVER_FORMAT = CSV
CLIENT_FORMAT = ScriptableObject
```

From reading the configuration file, the platform is informed that GO code is to be used for the server and C# code is to be used for the client. In this manner, the code generated by the platform can adapt the output serialised data object for use in either a server or a client. This configuration file also informs the platform that the serialised data object is to be exported in scriptable object format. Other desired export formats (such as JSON or CSV) are obtained by changing the FORMAT line accordingly and reloading the spreadsheet 200. One approach to create serialised data objects in all desired export formats is to have multiple pairs of the spreadsheet 200 and the configuration file, where each of the configuration files indicates one of the desired export formats for the FORMAT line. It will be appreciated that the configuration file can have additional lines listing other possible programming language export formats.

[042] After the serialised data objects are output in step 114, one or more validity checks on the data in the serialised data objects may be performed in step 116. For example, the serialised data objects may be checked to determine whether one or more of their attributes is empty, e.g. whether the "Example_String" field is empty; or whether there is a mismatch of data, such as whether the data type of the "Example_Class" field is correct. In addition, one or more of the analyses, described earlier to be done in step 106, may instead be done in step 116 instead. These include the nullity check and the duplicate field names check. Performing these checks in step 116 is advantageous to allow defects to be rectified before the serialised data objects are imported into the software game engine in step 118, to reduce troubleshooting.

[043] In step 118, the serialised data objects from step 114 are imported into a software game engine. The software game engine deserialises these serialised data objects in game with generated code while a game is running.

[044] Figure 8 shows generated code by the platform that manages retrieval of spreadsheets. The generated code allows the receipt of a search string comprising the name of the table 202, which is used to locate the table 202 from a plurality of tables saved in one or more folders. In this manner, the game developer can easily locate the spreadsheet 200 that contains the table 202.

[045] Returning to Figure 1, the platform executes, over the steps 106, 108, 110, 112 and 114, a method for creating serialised data objects from external data. The method sees the platform generating code 300 to analyse the table 202, comprising the external data organised in cells 206, to identify the type of data stored in each cell 206 to determine if the identified data types are parsable. The platform then generates, in response to the identified data types being parsable, further code 400 that when compiled outputs at least one serialised data object (522, 524; 622, 624; and 722, 724), wherein the data stored in each cell of the table is assigned to an attribute of the serialised data object (522, 524; 622, 624; and 722, 724). Finally, the platform retrieves an export format for the serialised data object (522, 524; 622, 624; and 722, 724) and compiles the generated further code 400 to output the serialised data object (522, 524; 622, 624; and 722, 724) in the retrieved export format.

[046] The steps 102 and 104 may also be part of the method that the platform executes to create serialised data objects from external data. That is, the method further comprises configuration of the spreadsheet 200, i.e. organising the external data into the cells 206 in accordance with an index 204 accompanying the table 202, the index 204 defining the data stored in each of the cells 206 in the table 202 through stating one or more properties of the stored data. The one or more properties specify one or more of, but not limited to: a field name for each of the cells 206, the type of data stored in the cell 206, and whether the data in the cell 206 is for use on a client or a server. The generated code 300 is configured to differentiate between the index 204 providing the data definitions and the table 202 comprising the external data.

[047] The method executed by the platform to create serialised data objects from external data may also perform one or more of the following actions.

[048] Analysing the index 204, when producing the further code 400 to create the serialised data object (522, 524; 622, 624; and 722, 724), to determine a combination of different data types for use as the attributes for the serialised data object.

[049] When analysing the table 202 to determine if the identified data types are parsable, perform one or more of the following: determine validity of data in each cell 206 against its respective one or more properties defined in the index 204 accompanying the table 202; a nullity check for cells 206 required to store data and generate an error message in response to detection of blankness; and a check for duplicate field names. An error message may be generated in response to the identified data types being determined to be imparsable.

[050] When processing a cell 206 in the table 202 containing multiple data sets, assign each of them to an attribute of the serialised data object (522, 524; 622, 624; and 722, 724).

[051] When locating the table 202 for analysis from a plurality of tables, receive a search string comprising the name of the table.

[052] Adapting the output serialised data object (522, 524; 622, 624; and 722, 724) for use in either a server or a client.

[053] The name for each of the attributes for the serialised data object (522, 524; 622, 624; and 722, 724) is obtained from a field name for the cell 206 providing the data for that attribute. The attributes include a unique identifier for the serialised data object. The export format may be one of, but not limited to, JSON, scriptable object, CSV.

[054] The data types that the generated code 300 is designed to parse comprise basic data types and custom data types used by a software game engine. Basic data types include any one or more of integer, string, float and Boolean. The custom data types include any one or more of: two and three dimensional vectors and are script specific.

[055] The above therefore discloses a platform that advantageously automates conversion of external data into in-game data objects, reduces human workload and errors, and improves work efficiency. A

variety of external data formats, serialization methods of data objects used by software game engines and language types of generated codes are supported.

[056] In the application, unless specified otherwise, the terms "comprising", "comprise", and grammatical variants thereof, intended to represent "open" or "inclusive" language such that they include recited elements but also permit inclusion of additional, non-explicitly recited elements.

[057] While this invention has been described with reference to exemplary embodiments, it will be understood by those skilled in the art that various changes can be made and equivalents may be substituted for elements thereof, without departing from the spirit and scope of the invention. In addition, modification may be made to adapt the teachings of the invention to particular situations, without departing from the essential scope of the invention. Thus, the invention is not limited to the particular examples that are disclosed in this specification, but encompasses all embodiments falling within the scope of the appended claims.

CLAIMS

1. A platform for creating serialised data objects from external data, the platform configured to generate code that

5 analyses a table, comprising the external data organised in cells, to identify type of data stored in each cell, wherein the data type is identified through reference to an index accompanying the table, the index defining properties of the data stored in each of the cells in the table, the table and the index being contained together in an external file;

determines if the identified data types are parsable;

10 generates, in response to the identified data types being parsable, further code comprising executable statements catered for the data in each of the cells, wherein the properties of the data defined in the index are referred to during the generation of the further code that when compiled outputs at least one serialised data object and wherein the data stored in each cell of the table is assigned to an attribute of the serialised data object; and

15 retrieves an export format for the serialised data object, wherein the platform is further configured to compile the generated further code to output the serialised data object in the retrieved export format.

2. The platform of claim 1, wherein the index is analysed when producing the further code to
20 create the serialised data object to determine a combination of different data types for use as the attributes for the serialised data object.

3. The platform of claim 1 or 2, wherein the properties comprise: a field name for each of the cells, the type of data stored in the cell, and whether the data in the cell is for use on a client or a server.

25 4. The platform of any one of the claims 1 to 3, wherein the generated code is configured to differentiate between the index providing data definitions and the table comprising the external data.

5. The platform of any one of the claims 1 to 4, wherein analysis of the table further comprises
30 determining validity of the data in each cell against its respective properties.

6. The platform of any one of the preceding claims, wherein analysis of the table further comprises performing a nullity check for cells required to store data and generate an error message in response to detection of blankness.

35 7. The platform of any one of the preceding claims, wherein analysis of the table further comprises performing a check for duplicate field names.

8. The platform of any one of the preceding claims, further configured to generate an error message in response to the identified data types being determined to be unparsable.
9. The platform of any one of the preceding claims, wherein the data types that the generated code is designed to parse comprise basic data types and custom data types used by a software game engine.
10. The platform of claim 9, wherein the basic data types include any one or more of integer, string, float and Boolean; and the custom data types include any one or more of two and three dimensional vectors and are script specific.
11. The platform of any one or more of the preceding claims, wherein a name for each of the attributes for the serialised data object is obtained from a field name for the cell providing the data for that attribute.
12. The platform of any one or more of the preceding claims, wherein the attributes include a unique identifier for the serialised data object.
13. The platform of any one or more of the preceding claims, further configured to process a cell in the table containing multiple data sets to assign each of them to an attribute of the serialised data object.
14. The platform of any one of the preceding claims, further configured to locate the table for analysis from a plurality of tables through receipt of a search string comprising the name of the table.
15. The platform of any one or more of the preceding claims, further configured to adapt the output serialised data object for use in either a server or a client.
16. The platform of any one of the preceding claims, wherein the export format is one of JSON, scriptable object, CSV.
17. The platform of any one of the preceding claims, wherein the export format is retrieved from a configuration file.
18. A method for creating serialised data objects from external data, the method comprising the steps of:
generating code to analyse a table, comprising the external data organised in cells, to identify type of data stored in each cell to determine if the identified data types are parsable, wherein the data

type is identified through reference to an index accompanying the table, the index defining properties of the data stored in each of the cells in the table, the table and the index being contained together in an external file;

5 generating, in response to the identified data types being parsable, further code comprising executable statements catered for the data in each of the cells, wherein the properties of the data defined in the index are referred to during the generation of the further code that when compiled outputs at least one serialised data object and wherein the data stored in each cell of the table is assigned to an attribute of the serialised data object;

10 retrieving an export format for the serialised data object; and
compiling the generated further code to output the serialised data object in the retrieved export format.

19. The method of claim 18, further comprising;
organising the external data into the cells in accordance with an index accompanying the table,
15 the index defining properties of the data stored in each of the cells in the table.