

stores the expression entered by the user so that case-sensitivity, spacing, and other particulars may be displayed back to the user as entered.

[0120] Turning now to **FIG. 4**, a diagram illustrating the execution of an exemplary byte code is shown. In this example, the user has specified a function in the form of an expression. The expression is subsequently parsed to generate the byte code, which is a high level set of instructions representing the desired operations and which is executed at run-time. The execution of the byte code is diagramed in **FIG. 4**.

[0121] **FIG. 4** shows the execution of an exemplary `VcStmtCreateShape` byte code **302**. The byte code **302** creates an instance of a particular shaped object when executed and records a tokenized name for the object for a hash table lookup. One exemplary implementation pseudocode for the construction of the object shape to be generated by the byte code **302** is shown as block **310**.

[0122] In the implementation pseudocode block **310** for constructing the shape of the object, when a `VcStmtCreateShape`'s `Perform()` method is called, it calls a factory object's `CreateShape()` method to create a new shape object and then stores a pointer to the object in a hash table owned by a context object for later lookup when setting properties. Finally, the method releases the reference to the shape established during its creation.

[0123] The byte code **302** has an `m_objectId` property which is a member of the standard `CString` class **300** to provide storage for the tokenized name of the shape. The byte code **302** has a `m_factory` property which is a member of the abstract base class `VcShapeFactory` **304** which is responsible for creating a shape within a scene. Derived classes of `VcShapeFactory` **304** must override the pure virtual `CreateShape()` method in order to create a particular class of shape and add it to a canvas display list.

[0124] `VcShapeFactory` **304** creates an instance of a shape object which is derived from the abstract base class `VcShape` **306**. `VcShape` **306** is an abstract base class for all shape objects. It provides a transaction-oriented, polymorphic interface for setting shape properties. Derived classes of `VcShape` are responsible for implementing rendering and hit-detection support.

[0125] **FIG. 4** also shows an abstract class `VcStmtSetProperty` **320**, which is an abstract class for byte code statements to set the value of a shape's property when executed. Derived classes of `VcStmtSetProperty` **320** represent each data type. The abstract class `VcStmtSetProperty` **320** stores the tokenized name of the shape and a set of the property identities necessary to uniquely address the property. `VcStmtSetProperty` **320** has a property `m_objectId` which is a member of the `CString` class **322** and an `m_propPath` property which is a member of class `VcPropertyPath` **324**. `VcPropertyPath` **324** stores an identifier for a property that is unique within the shape's set of properties, and provides a pointer, `m_next`, to a linked list `VcPropertyPath` to support aggregated shapes.

[0126] `VcStmtSetProperty` **320** in turn is inherited by `VcstmtSetStringProperty` **326** which sets the value of a shape's property to a string value, and which contains a pointer to an abstract string function `VcSftn` **328** for evaluating the property value before setting the shape's property.

`VcSftn` **328** is derived from `VcFunction` which evaluates to a string for a given context and the function is the root element for a parsed expression tree containing the parsed elements of the function.

[0127] One implementation of the `VcStmtSetStringProperty` byte code **326** is shown in block **330**. In this implementation for setting a shape's property, when the `VcStmtSetStringProperty`'s `Perform` method is called, it retrieves the shape from the context's hash table. It then evaluates the string function with the given context and passes the result to the shape via the property path.

[0128] Turning now to **FIG. 5**, a diagram illustrating a sample parsed property function is shown. The function in this case is the `VcStmtSetStringProperty` function **350**. The function **350** has an object identification value "Text1"**352**, which is a member of the class `CString`. This is the tokenized name of the shape object containing the property to be set. `VcStmtSetStringProperty` **350** is a sample byte code statement which sets the value of the `Text` property owned by the object `Text1` to the calculated value resulting from appending `last_name` to `first_name`, separated by a space character. In this example, `m_objectId` points to a string identifying the target object, `Text1`; `m_propPath` points to the address of the property to be set within the text object; and `m_ftn` points to a string function which evaluates to the property's value.

[0129] `VcStmtSetStringProperty` **350** has a property path whose values are set in Block **354**. Finally, `VcStmtSetStringProperty` **350** has been designated to execute a `VcSftnConcatenate` function **360**. `VcSftnConcatenate` **360** is a string function derived from `VcSftn` which concatenates the results of two member string functions, `m_ftn1` (`first_name+" "`) and `m_ftn2` (`last_name`).

[0130] Within the block **360**, two functions are further specified. Traversing down the left branch of the tree, a `VcSftnConcatenate` function **362** is designated. `VcSftnConcatenate` **362** is responsible for evaluating the expression (`first_name+" "`) using its two member string functions, `m_ftn1` and `m_ftn2`. Within the block **362** are two additional functions, a `VcSftnLookup` function **364**, a string function responsible for looking up the current value of the identifier stored in `m_ref` ("first_name"), and a `VcSftnConstant` function **370**, a string function responsible for storing a constant string value.

[0131] The evaluation of the `Lookup` function **364** is determined by character string **366**, which is a standard string class to provide storage for the name of the identifier, "first_name". The evaluation of the constant function **370** results in a character string which is a space (the string constant " ") in block **372**.

[0132] Traversing down the right side of the tree from the `VcSftn concatenate` function **360**, a `VcSftnLookup` function **380** is evaluated. `VcSftnLookup` **380** is a string function responsible for looking up the current value of the identifier stored in `m_ref` ("last_name"). The evaluation of the `VcSftnLookup` function **380** results in a character string.

[0133] An object model of the code generation process from a scene is illustrated in more detail in **FIG. 6**. The scene contains a list of drawing nodes representing the code generators for each shape and logic element within the scene. An object `VcScene` **400** defines a drawing layer or canvas for all graphical objects displayed in the viewing