: **MOTEL** : Model Object Tuple Expression Language

: **OBJECT**, 1 :

Model Object Tuple Expression Language (MOTEL) is an object oriented name-value information management language. MOTEL objects are used to define a document’s structure and provide semantics of data within document instances. MOTEL object declarations have three parts: a name, a value and member objects.

The object name is used by the MOTEL processor to reference an object instance to access or alter its value or members.

The object’s value component holds data. Each object is an array of zero or more values. The object’s values are referenced in MOTEL script via an array index. Objects may have processing rules that operate upon the object value. MOTEL data processing rules may be defined for an object with an MOTEL pattern or for a document in MOTEL script (both defined elsewhere).

Every object has zero or more subordinate member objects. Subordinate member objects occur within the scope of a parent object and subsequently its ancestral object hierarchy. Objects may be referenced using a fully qualified hierarchical scope. Object hierarchies are known as complex data types or data structures.

MOTEL supports multiple object inheritance, partial object inheritance and object extension. An MOTEL object may include some or all of the members of an inherited object.

Character

These are one-to-one mappings of characters from external character sets to MOTEL objects. Characters are the most fundamental MOTEL objects, providing a raw mechanism by which to recognize character values within MOTEL documents.

Character objects are only used to declare Graphemes.

Grapheme

These are many-to-one mappings of Character objects to MOTEL objects. Graphemes are the second most fundamental MOTEL objects, providing a mechanism by which to declare Character objects as a logical character. When processing MOTEL documents only the logical character values represented by Graphemes are used.

Grapheme objects are only used to define Morphemes.

Morpheme

These are many-to-one mappings of Grapheme objects to MOTEL objects. Morphemes are the third most fundamental MOTEL objects, providing a processing mechanism to define Grapheme object classes and MOTEL parser control objects.

Document

A collection of MOTEL object definitions and comments

: **CHARACTER** :

A character is an MOTEL object that contains a byte pattern that represents an atomic MOTEL lexical token (e.g. number, letter, punctuation or other symbol). Unicode is an example of a character set that can be used to define MOTEL lexical tokens. The name of the character within its source character set is should be used for its corresponding MOTEL character definition (e.g. The Unicode character named DIGIT ZERO will be named DIGIT ZERO as an MOTEL lexical token).

CHARACTER objects may be defined using an ASCII character, EBCDIC character, Unicode character or numeric value. Binary, octal, decimal and hexadecimal numeric values are supported.

: **GRAPHEME** :

MOTEL GRAPHEME objects are solely defined using MOTEL CHARACTER objects. A GRAPHEME object is a construct that groups CHARACTER objects into an equivalence class. Each equivalence class may map one or more CHARACTER objects to a GRAPHEME object. GRAPHEME objects are used by the MOTEL parser to process MOTEL files.

To simultaneously utilize an accented and an unaccented character set, the accented character and a corresponding unaccented character could be mapped to the same grapheme (e.g. LARGE A could be mapped to LATIN CAPITAL LETTER A WITH GRAVE and LATIN CAPITAL LETTER A. In this case the parser would recognize LATIN CAPITAL LETTER A WITH GRAVE and LATIN CAPITAL LETTER A as LARGE A when processing a MOTEL file).

A GRAPHEME object may be composed of one or more GRAPHEME objects in a set (e.g. NEW LINE could be composed of a CARRIAGE RETURN and a LINE FEED). When parsed, a grapheme that is a superset of another grapheme has precedence (e.g. NEW LINE will be returned by the parser when a CARRIAGE RETURN immediately followed by a LINE FEED is encountered. Additionally, the parser will not return the CARRIAGE RETURN or LINE FEED graphemes when these are combined into a NEW LINE).

: **MORPHEME** :

A morpheme is an MOTEL object that uses grapheme patterns to represent atomic MOTEL processing tokens (e.g. grapheme classes, MOTEL script or object masks).

Grapheme class objects are an enumerated list of values. These values exist within the object as an array and may be accessed by index, sorted and searched.

Script morphemes are operated upon by the script processing engine and must adhere to MOTEL script syntactic rules.

Object masks are defined to specify the type, input format or output format of an object value. Object masks are automatically executed upon value access or applied to objects during MOTEL script execution.

: **MOTEL DOCUMENT**, 8 :

MOTEL documents are composed of scope declarations, object declarations, schema references and comments.

The contents of an MOTEL document are declared using MOTEL semantic tokens. Each semantic token has a reserved object name that the MOTEL parser recognizes. The MOTEL semantic tokens are known as parser graphemes. Parser graphemes are defined within the standard *grapheme.motel* Grapheme Definition File under the **MOTEL**.GRAPHEME scope. The parser morphemes defined within the standard *morpheme.motel* Grapheme Definition File may be overridden via custom grapheme definitions.

The first parser morpheme is DECLARATION DELIMITER (fully specified **MOTEL**.MORPHEME.DECLARATION DELIMITER). The DECLARATION DELIMITER morpheme marks the beginning and ending of an object declaration. Object declarations include an object name optionally followed by any value processing control parameters (e.g. there must be a DECLARATION DELIMITER before and after the object name and the value processing control values). The default for the DECLARATION DELIMITER is COLON.

The second parser morpheme is MEMBER DELIMITER (fully specified **MOTEL**.MORPHEME.MEMBER DELIMITER). The MEMBER DELIMITER morpheme is used to delimit an object member within an object name (e.g. MOTEL.GRAPHEME where the period serves the role of the member delimiter). By convention the default for the MEMBER DELIMITER is FULL STOP.

The third parser morpheme is LIST DELIMITER (fully specified **MOTEL**.MORPHEME.LIST DELIMITER). The LIST DELIMITER morpheme is used to delimit the object name from any processing control values. When no processing control values are present the LIST DELIMITER is omitted. The default for the LIST DELIMITER is COMMA.

The fourth parser morpheme is IMPORT (fully specified **MOTEL**.MORPHEME.IMPORT). The IMPORT morpheme is used to instruct the parser to import declarations from an MOTEL file. The default value of IMPORT is ASTERISK.

Within the standard *morpheme.motel* Morpheme Definition File are composite semantic tokens know as Morphemes.

The parser morpheme LAYOUT defines a set of characters that are treated solely as visual formatting (whitespace) characters. When parsed these formatting characters are discarded before and after object names and are compressed to a single space within object names. Within object values leading and trailing formatting characters are discarded and embedded formatting characters are preserved.

To cause the parser to specially process or preserve formatting characters within an object value, an MOTEL Pattern Language (MPL) mask may be defined for the object. MPL pattern masks can specify an object’s data type, input format, validation rule and output format.

Processing control parameters tell the MOTEL engine how to process the value portion of an MOTEL object declaration. Processing control parameters are separated by the LIST DELIMITER grapheme.

The first processing control parameter is the DECLARATION DELIMITER count. This parameter informs the MOTEL parser to include a number of DECLARATION DELIMITER graphemes as part of the object value (e.g. ***:XXX,3:A:B:C:D*** – where DECLARATION DELIMITER is CHARACTER.COLON, the parameter value of 3 instructs the MOTEL parser to include the colons between A and B, B and C, C and D as part of the value of object XXX).

The DECLARATION DELIMITER count may be set to a value of UNLIMITED to represent a streaming member. The default value of UNLIMITED is ASTERISK (e.g. ***:XXX,\*:…streaming content…)***. Streaming members exist to support arbitrarily long values where the content is not known in advance or would be inconvenient to determine the DECLARATION DELIMITER count (e.g. video content). The end of the document (e.g. stream close) is treated as the end of the member value. Streaming members are typically processed as received. Each document can only have one streaming member. A streaming member must be the last member of a document. To handle multiple streaming members, a parent MOTEL document is created that references (e.g. via an HRR or URL) its child MOTEL documents, each of which is processed potentially via an independent thread of execution and/or (socket or stream).

The object name may be omitted from an MOTEL declaration to insert a comment within an MOTEL file. Any text following the unnamed object (as represented by a DECLARATION DELIMITER grapheme pair) will be parsed and validated but not loaded into an MOTEL object (e.g. *:,1: Annotation: Vehicle 1 and 2*). Note: processing parameters may be applicable to comments.

Namespaces are used to scope member objects within a collection. A named object without a value declares a namespace (e.g. the :CHARACTER: declaration at the beginning of the default MOTEL character.MOTEL file). All objects declared after a namespace declaration are members of the namespace and may be referenced using MEMBER DELIMITER notation (e.g. CHARACTER.NUMBER SIGN).

Namespace objects are able to hold values by declaring one or more values for the name space object. The namespace object value instances must be immediately children of the namespace, often defined immediately following the namespace declaration.

The first namespace object within an MOTEL file is considered to be at the root of the document (e.g. the first namespace is not in the scope of any other namespace). Namespace objects can be independent from or subordinate to a parent namespace.

Scope declarations are used to define document structure. Scope declarations are object declarations without a value. All objects defined beneath a scope declaration are contained within its scope.

MOTEL files are a collection of sequenced references and object definitions where antecedent references and object definitions are available to be used to declare subsequent objects within an MOTEL file.

Documents define objects within scopes. All documents have an initial scope of document level, denoted by a fully qualified object name of only a DECLARATION DELIMITER (e.g. .ORDER references the order object at the document level).

By convention, each MOTEL file defines an object at the document level that is named the same as the MOTEL file. This member sets the scope for (contains) objects that pertain to the primary purpose of the MOTEL file. Other members may be also declared at the document level to support or qualify the primary purpose of the document (e.g. reusable schema declarations, formatting, MOTEL script or security information).

Objects may be declared using a relative or fully qualified scope.

Antecedent objects may be inherited by reference and extended by subsequent object declarations within a document.

The IMPORT parser directive is used to import external document declarations. The IMPORT parser directive instructs the parser to open the MOTEL file reference contained within the object value to import members within the current document. The IMPORT parser directive occurs within the name portion of an MOTEL declaration.

To import all objects to a document the IMPORT parser directive is placed at the beginning of the object name.

:\*,1: http://www.speakhuman.org/ws/something.motel

To import an object to a document the IMPORT parser directive is placed at the beginning of the object name, followed by a MEMBER DELIMITER, followed by the name of the member to import.

:\*.*THING*,1: http://www.speakhuman.org/ws/something.motel

Fully qualified scope references may also be used to set the location at which imported members are imported within a document.

:*SCOPE*.\*,1: http://www.speakhuman.org/ws/something.motel

or

:*SCOPE*.\*.*THING*,1: http://www.speakhuman.org/ws/something.motel

Members of multiple import files may be combined within a single scope using fully qualified scope references.

:*SCOPE*.\*,1: http://www.speakhuman.org/ws/something.motel

and

:*SCOPE*.\*,1: http://www.speakhuman.org/ws/something.else.motel

All members from something.motel and something.else.motel would be included within a shared scope in the order that the members were imported.

MOTEL script can reference an object using fully qualified scope references.

The following MOTEL document excerpts are semantically identical:

|  |  |  |
| --- | --- | --- |
| **Unorganized** | **Grouped, Full Scope** | **Grouped, Relative Scope** |
| :OBJECT 1:AB  :OBJECT A:  :OBJECT 2:CD  :OBJECT B:  :OBJECT 3:EF  :OBJECT 4:GH  :.OBJECT A.OBJECT C:  :OBJECT 5:IJ  :.OBJECT D:  :OBJECT D:KL  :OBJECT D:MN  :OBJECT E:  :OBJECT 6:OP  :.OBJECT A.OBJECT B:  :OBJECT B:QR  :OBJECT 4:ST  :OBJECT F:  :OBJECT 7:UV  :.OBJECT A.OBJECT C:  :OBJECT 8:WX  :.OBJECT D.OBJECT E:  :OBJECT 9:YZ  :.OBJECT 1:ab | :OBJECT 1:AB  :OBJECT 1:ab  :.OBJECT A:  :.OBJECT A.OBJECT 2:CD  :.OBJECT A.OBJECT B:  :.OBJECT A.OBJECT B:QR  :.OBJECT A.OBJECT B.OBJECT 3:EF  :.OBJECT A.OBJECT B.OBJECT 4:GH  :.OBJECT A.OBJECT B.OBJECT 4:ST  :.OBJECT A.OBJECT B.OBJECT F:  :.OBJECT A.OBJECT B.OBJECT F.OBJECT 7:UV  :.OBJECT A.OBJECT C:  :.OBJECT A.OBJECT C.OBJECT 5:IJ  :.OBJECT A.OBJECT C.OBJECT 8:WX  :.OBJECT D:  :.OBJECT D:KL  :.OBJECT D:MN  :.OBJECT D.OBJECT E:  :.OBJECT D.OBJECT E.OBJECT 6:OP  :.OBJECT D.OBJECT E.OBJECT 9:YZ | :OBJECT 1:AB  :OBJECT 1:ab  :OBJECT A:  :OBJECT 2:CD  :OBJECT B:  :OBJECT B:QR  :OBJECT 3:EF  :OBJECT 4:GH  :OBJECT 4:ST  :OBJECT F:  :OBJECT 7:UV  :.OBJECT A.OBJECT C:  :OBJECT 5:IJ  :OBJECT 8:WX  :.OBJECT D:  :OBJECT D:KL  :OBJECT D:MN  :OBJECT E:  :OBJECT 6:OP  :OBJECT 9:YZ |

* OBJECT A is a scope object in the scope of the document root
* OBJECT B is a scope object and a single value data object in the scope of OBJECT A
* OBJECT C is a scope object in the scope of OBJECT A
* OBJECT D is a scope object and a multiple value data object in the scope of the document root
* OBJECT E is a scope object in the scope of OBJECT D
* OBJECT F is a scope object in the scope of OBJECT B
* OBJECT 1 is a multiple value data object in the scope of the document root
* OBJECT 2 is a single value data object in the scope of OBJECT A
* OBJECT 3 is a single value data object in the scope of OBJECT B
* OBJECT 4 is a multiple value data object in the scope of OBJECT B
* OBJECT 5 is a single value data object in the scope of OBJECT C
* OBJECT 6 is a single value data object in the scope of OBJECT E
* OBJECT 7 is a single value data object in the scope of OBJECT F
* OBJECT 8 is a single value data object in the scope of OBJECT C
* OBJECT 9 is a single value data object in the scope of OBJECT E

: **Character Definition File** :

The Character Definition File is used by the MOTEL parser to declare the numeric codes/byte patterns for character objects. By convention, character names closely match the names of the characters as defined by the character encoding (e.g. The Unicode character named COLON has the value U+003A[[1]](#endnote-1)).

The character set used to create the Character Definition File must correspond to the parser’s recognized character set(s). For instance a parser built for an AS/400 would use EBCDIC for its Character Definition file, whereas a UNIX parser would use ASCII. The parser could support multiple character sets, being able to either automatically detect or allow the user to specify the character set via an execution parameter.

The Character Definition File is in MOTEL format and must use the MOTEL default reserved grapheme definitions (e.g. DECLARATION DELIMITER, MEMBER DELIMITER and LIST DELIMITER).

Within a character definition the value type may be specified using a type specifier. The type specifier occurs immediately after the object name, separated from the object name by the LIST DELIMITER. The type specifier is a single character of one of the following; B for binary, O for octal, D for decimal, H for hexadecimal, U for Unicode, A for ASCII and E for EBCDIC. When no type specifier is provided a value of the type of the file character set is assumed.

: **Grapheme Definition File** :

The Grapheme Definition File maps character definitions to graphemes. A grapheme is normally comprised of a single character and is named to match the **MOTEL**.CHARACTER from which it is defined.

Meta-characters

: **Parser** :

Upon start up the parser may load a character definition file to declare the character list that the parser will accept. Character codes that do not conform to this list are illegal and will cause the parsing of a document to fail.

Alternatively, the parser may be implemented with a default character set that is used when no character definition file is provided.

: **Type** : (1) Has a default character set and does not support a character definition file

: **Type** : (2) Has a default character set and supports a character definition file (the file character definitions override the default character set member definitions)

: **Type** : (3) Has no default character set and requires a character definition file

: **Script Engine** :

: **Expression Engine** :

1. <http://unicode.org/Public/UNIDATA/UnicodeData.txt> [↑](#endnote-ref-1)