UGH: Utilisable Genuine Homonym

Author:	Bryan Jurish
Date:	April 19, 2000
Version:	1.6.4
Platform:	Delphi / SQL
Copyright Policy:	Pretty relaxed, actually.

Contents

Chapter 1

Administrivia

1.1 Legal Bits

1.1.1 Legal issues for the C part of the Regular Expression package

Beware that some of this code is subtly aware of the way operator precedence is structured in regular expressions. Serious changes in regular-expression syntax might require a total rethink.

1.1.2 Legal issues for the Delphi part of the TGrep Regular Expression package

NOTE: The TGrep package as distributed with UGH has been altered from its original form. The original, unaltered, TGrep package is distributed, within its own subdirectory, along with UGH. Specifically, in the file `Grep.pas`, the property `SubExp[index: TSubExpRange]` has been changed from "published" to "public" status, because Delphi Versions ¿= 4.0 complained otherwise.

1.1.3 Legal issues for the UGH package

- Parts Copyright ©1986 by University of Toronto, see Section 1.1.1 for legal information concerning these.
- Parts Copyright ©1998 by Gabriele Carioli, see Section 1.1.2 for legal information concerning these.

All other parts Copyright ©1998-2000 by Bryan Jurish, <jurish@ling.uni-potsdam.de>.

Permission is granted to anyone to use this software for any purpose on any computer system, and to redistribute it freely, subject to the following restrictions:

1. No responsibilty is taken on the part of the author or on the part of the author's employers for damages (mental, phyiscal, or otherwise) resulting from the use of this software, even if these damages result from defects in the software.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software.

Beware! Undocumented features!

1.2 Acknowledgements

Many thanks to Gabriele Carioli for the TGrep wrapper, and to Henry Spencer, for the Win32 regexp.dll. Oodles of gratitude also to any and all who put up me with during UGH's construction.

1.3 UGH: Statement of Purpose

UGH, the Utilisable Genuine Homonym, is a byproduct of its author's quite possibly somewhat twisted belief that moving data around is an action which ought to be accomplishable by the presence of some more data indicating exactly how this ought to be done.

Due to incompatibilities between the diverse implementations of SQL existing in the world, it can be considered desirable to avoid the use of a specific implementation's peculiarities for purposes of simple data extraction, particularly when the data to be extracted by a given task or program is not itself restricted to a particular SQL dialect. UGH is an attempt to overcome these problems by means of reducing a set of partial SQL queries (known as *UGH Specifications* or simply Specs) into a (hopefully) well-formed SQL query, according to ANSI SQL-1. UGH is currently implemented in Borland Delphi 5.0 for Win32, although it should run in Delphi versions 4 and 3, as well, and possibly even in Delphi 2.

Chapter 2

Intro to UGH

2.1 Overview

UGH, the Utilisable Genuine Homonym, is a software development package which makes use of a user-defined *database specification*[1] to perform a dynamic mapping from a sequence of abstract *search criteria*[2] to a well-formed SQL query. The system was first developed in early 1998 at the University of Potsdam, Institute for Linguistics as part of a project to create an intuitive graphical user interface for querying a lexical database. The UGH system was written and is currently supported by Bryan Jurish, a student assistant at the University of Potsdam.

In its task to provide an intuitive interface to an embedding application as well as a method of specifying a target database's structure independently of any embedding application, UGH first *compiles*[3] a database specification into an UGH "Engine", which can then perform mappings from the embedding application's abstract search criteria to SQL queries. Each mapping from a set of search criteria to a single SQL query is referred to as an UGH

[1]See Section 2.3 for details on database specifications.
[2]See Section 2.2 for details on search criteria.
[3]See Section 2.4.1 for details on the compilation process.

matching[4].

2.2 Search Criteria

The *search criteria* from which an UGH Engine endeavors to construct an SQL query are simply "label=value" pairs, and are passed to UGH by an embedding application in the form of a Delphi TStrings object. The value part of a search criterium may be nil[5], an atomic (literal) value, or a complex value built from atomic values and Engine-specific characters representing the logical connectives "and" and "or" (by default "&" and ";" respectively). A search criterium's value may also contain references to variable values, as well as to the values of other search criteria.[6] Multiple sets of disjoint search criteria[7] are also supported.

NOTE: In older parts of this document, the terms "SearchSet" and "KeyVals" may be used to refer to a set of search criteria, "key" may be used to refer to the label of an individual criterium, and "keyval" may be used to refer either to a search criterium as a whole or to its value.

2.3 Database Specifications

An UGH database specification is the source from which an UGH Engine is compiled. In older versions of UGH, a database specification could either be read from an ASCII source file in the format described in Section 3, or could be located in an external database in the format described in Section 4.

[4] See Section 2.4.2 for details on the matching process

[5] The special string *VarNull*, which defaults to "-", is interpreted in many cases differently by the UGH Engine than other criterium values.

[6] See Section 2.4.3 for details on UGH's methods of referencing and interpolating variable values.

[7] Multiple sets of search criteria are passed to UGH in the form of a TStringListList object.

An UGH database specification is composed of a set of individual *table-*, *projection-*, *selection-*, and *meta-* specifications, and associates with each search criterium label at most one projection- and at most one selection-specification. These can then be directly mapped to SQL fragments (in the SELECT and WHERE clauses, respectively). Inner join clauses in the SQL WHERE clause are automatically constructed based on the table-specifications indicated by the selection- and projection-specifications activated by a given set of search criteria. The procedural entities known as "meta-specifications", the interpolation of global and local variables, and regular transductions provide mechanisms for dealing with ambiguous or interdependent search criteria.

NOTE: In older portions of this document, the term "key specification" may be used to refer to a selection specification, and the term "col specification" may be used to refer to a column specification. Similarly, the abbreviation "specs" may be used to refer to an UGH database specification. Don't let it disturb you.

2.4 Processes

2.4.1 Compilation

An UGH Engine must first be compiled from a source file[8] before it may be used to generate SQL queries. Compilation can be easily accomplished by means of a Delphi TUGHSource component, or a triple of TUGHEngine, TUGHSource, and TUGHDatabase components. The "Compile" and "CompileLocal" methods of TUGHSource or TUGHDatabase will serve to compile the associated TUGHEngine object. Note that a TUGHSource object's "FileName"[9] property must be set before calling "Compile" or "CompileLocal".

[8] or database, pre-v1.6.4

[9] or "Database", pre-v1.6.4

2.4.2 Matching

The UGH process of mapping a set[10] of search criteria to an SQL query is known as matching, and performs for each set of search criteria the following steps, which collectively are referred to as "Context Matching".

1. **Preliminary Matching**
   Meta specifications with "on" subspecifications matching *SearchSetBegin* are executed.

2. **Meta Matching**
   Meta specifications with "on" subspecifications matching some non-ignored label of the search set are executed.

3. **Extended Matching**
   Each search criterium's value is passed to the "extended specification"[11] whose name matches its label (if one exists), as the value of *DefaultValueReference*. Logical connective-characters are expanded by selection-specifications, and each individual value is bound in turn to *DefaultValueReference*. Interpolation of these and of other values is performed.

4. **Post-Extended Matching**
   Meta specifications with "on" subspecifications matching *SearchSetEnd* are executed.

5. **Interpolation**
   Full interpolation is performed on any interpolated strings generated by the current search set.

After all sets of search criteria have undergone the context matching process, the UGH Engine constructs an SQL query by the following means:

1. **Join Construction**
   Join-clauses are constructed for the table specifications whose names

[10] or set of sets

[11] Extended specifications include projection and selection specifications.

match the interpolated “table” subspecifications of the activated projections and selections. Due to $\mathcal{NP}$ problems with Steiner trees, all tables must be (at least “virtually”) adjacent. Unjoinable tables will produce an UGH error.

2. **Priority Sorting**
   The activated projections and selections are sorted in descending order of their interpolated “priority” subspecifications.

3. **Clause Construction and Gathering**
   The separate SQL Clauses[12] are constructed and gathered into the SQL TStrings object requested.

Matching is typically invoked by one of the following top-level TUGHDatabase methods:

```
GenerateQuery(KeyValList: TStringListList);
GenerateLocal(KeyValList: TStringListList; LocalSQL: TStrings);
GenerateQuerySimple(KeyVals: TStrings);
GenerateLocalSimple(KeyVals, LocalSQL: TStrings);
```

2.4.3 Interpolation

The UGH Engine performs string interpolation on nigh unto every string that crosses its path, including (but not limited to) criterium values, as well as “in”, “as”, “query”, “case”, and “do” subspecifications of projection, selection, and meta specifications. All interpolation is performed at matchtime, and all variable bindings - with the exception of global variables - are local to the search set being matched when they were created. Values of variables, search criteria, and the result of the execution of a meta specification may be interpolated into any of the above strings. Details on string interpolation syntax and processes may be found in Section 3.4.

[12]Currently, the SELECT, FROM, WHERE, and ORDER BY clauses are implemented.

Chapter 3

UGH File Syntax

3.1 Notes on the Syntax Notation

The syntax notation conventions used in the following sections are described below.

- `Fixed-Width Text` will be used to designate literal character strings.
- *Slanted Text* will be used to indicate (hopefully mnemonic) labels for nonterminal symbols.
- Two colons followed by an equals sign (::=) indicate the definition for a nonterminal symbol.
- A vertical bar (|) indicates a logical disjunction of the listed possibilities.
- Curly brackets },{ logically group disjunctive possibilities.
- Square brackets],[indicate an optional element.
- A dot (.) indicates any character.

- An asterisk (*) indicates 0 or more repetitions of the preceeding expression.
- The nonterminal symbol $< nl >$ indicates a newline or carriage return.
- The nonterminal symbol $< space >$ indicates a space character.
- The nonterminal symbol $< tab >$ indicates a tab character.

3.2 UGH Declarations

3.2.1 Syntax

$$UGH-Declaration \quad ::= \{ \begin{array}{l} Comment \mid \\ Blank-Line \mid \\ UGH-Specification \ \} \end{array}$$

$$UGH-Specification \quad ::= \{ \begin{array}{l} Meta-Specification \mid \\ Table-Specification \mid \\ Selection-Specification \mid \\ Projection-Specification \ \} \end{array}$$

$$Whitespace \quad ::= \quad < space > \ \mid \ < tab > *$$

$$Comment \quad ::= \quad Whitespace * \texttt{\#} .* \ < nl >$$

$$Blank-Line \quad ::= \quad Whitespace* \ < nl >$$

3.2.2 Gratuitous Commentary

All Indicated newlines are literal. Newlines may be escaped by preceeding the literal newline or carriage return with a backslash (\). Single-line comments and blank lines are allowed. Any whitespace at the beginning or end of a line is removed. For anything else, you're at your own risk.

3.3 UGH Meta Specifications

3.3.1 Meta Syntax

$$
\begin{aligned}
Meta - Specification \ ::= \ & \texttt{META:}MetaName \ <nl> \\
& [\texttt{IGNORE:}IgnoreVal \ <nl>] \\
& [\texttt{DISMISS:}DismissVal \ <nl>] \\
& [\texttt{ON:}MatchVal \ <nl>] \\
& [\texttt{DO:}ActionSubSpecification]* \\
& [\texttt{EXEC:}ExecMetaName \ <nl>] \\
& [\texttt{CALL:}SubContext \ <nl>]*
\end{aligned}
$$

$$MetaName \ ::= \ LiteralString$$

$$IgnoreVal \ ::= \ LiteralString$$

$$DismissVal \ ::= \ LiteralRegularExpression^{1}$$

$$
\begin{aligned}
MatchVal \ ::= \{ \ & VarNull \ | \\
& SearchSetEnd \ | \\
& SearchSetBegin \ | \\
& LiteralRegularExpression \ \}
\end{aligned}
$$

$$ExecMetaName \ ::= \ InterpolatedString$$

$$ExtendedName \ ::= \ InterpolatedString$$

$$SubContext \ ::= \ SubContextName\texttt{=}SubContextValue$$

$$SubContextName \ ::= \ InterpolatedString$$

$$SubContextValue \ ::= \ InterpolatedString$$

$$LiteralRegularExpression \ ::= \ LiteralString$$

$$LiteralString \ ::= \ .*^{2}$$

[1] For details on the regular expressions used by UGH, consult the documentation of the `regexp.dll` package, or the `egrep`(1) manpage.

$VarNull$ ::= `-`

$SearchSetEnd$ ::= `->`

$SearchSetBegin$ ::= `<-`

3.3.2 Meta Invocation

An UGH Meta Specification may be invoked in any one of the following ways:

1. The Meta may be invoked during the initial phase of the context-matching process for a search set by virtue of the *MatchVal* in the Meta's `ON:` Subspecification. Possible causes for this type of meta-invocation include the follwing:

   (a) The Meta's *MatchVal* is equal to *SearchSetBegin*.

   (b) The Meta's *MatchVal* is a Regular Expression which matches the label of some criterium present in the search set, the value of which is neither literally equal to the Meta's *IgnoreVal* nor a positive match with the Meta's regular expression *DismissVal*[3]. If the Meta's *MatchVal* is equal to *VarNull*, no effort to match the Meta to the criteria in the search set is made.

   (c) The Meta is invoked indirectly, either through the string interpolation mechanism or through an Exec subspecification of some other Context Specification which is evaluated during this phase.

2. The Meta is invoked during string interpolation, if its name occurs as the *MetaName* part of a *DelegateExpression* within some interpolated string.

[2]See Section 3.4 on string interpolation for details on escaping special characters in literal and interpolated strings, and for an extended syntax of UGH's interpolated strings.

[3]A criterium present in a given search set whose value is either literally equal to a Context Specification's[4]*IgnoreVal* or which positively matches the regilar expression in that Context Specification's *DismissVal* will be referred to in the future as "ignored with respect to" that Context Specification.

[4]Context specifications include meta, projection, and selection specifications.

3. The Meta may be invoked during any phase of context matching, if its name occurs as *ExecMetaName* within an Exec Subspecification of another Context Specification which is evaluated during this phase and whose value is nonignored with respect to the Meta.

4. In the final phase of context matching for a search set, if the Meta's *MatchVal* is equal to *SearchSetEnd*.

3.3.3 Meta Execution

When a Meta is invoked and its value is non-ignored, the Meta is said to execute, and the following occurs:

1. The Meta's Return Value is initialized to be its Parameter Value. A Meta invoked by virtue of a match with some criterium in a search set or by some Exec Subspecification carries as a Parameter Value the value of the invoking criterium or Specification. This value is available within Meta-internal interpolated strings as *DefaultValueReference*.

2. The Meta's Action Subspecifications[5] are evaluated in the order in which they occured in the source file. See Sections 3.3.4 and 3.3.5 for details.

3. If another Meta is named in the Exec Subspecification of the Meta being executed, then the Meta named *ExecMetaName* is executed.

4. If any Call Subspecifications are present, then context matching is recursively invoked on a Search Set composed of the given Call Subspecifications and their values, which are first interpolated with respect to the current Search Set and *DefaultValueReference*.[6]

5. The Meta returns the value of *DefaultValueReference*.

[5] also referred to as `DO:` Clauses

[6] This procedure is not for the weak of heart: handling of recursively generated Search Sets needs some work.

3.3.4 Action Syntax

$ActionSubSpecification$ ::= $VarName$ $ActionClause$

$VarName$ ::= $InterpolatedString$

$ActionClause$ ::= { $AssignmentClause$ |
$GlobalizeOperation$ }

$AssignmentClause$ ::= $AssignmentOperator$
$ValueExpression$

$AssignmentOperator$ ::= { $RegExpAssign$ |
$GlobalAssign$ |
$ClobberAssign$ |
$NoClobberAssign$ }

$ValueExpression$ ::= { $InterpolatedString$ |
$RegExpTransduction$ }

$GlobalizeOperation$ ::= { $ClobberGlobalize$ |
$NoClobberGlobalize$ }

$ClobberGlobalize$ ::= `:!`

$NoClobberGlobalize$ ::= `?!`

$RegExpAssign$ ::= `~=`

$ClobberAssign$ ::= `:=`

$NoClobberAssign$ ::= `?=`

$GlobalAssign$ ::= `!=`

$RegExpTransduction$ ::= $SearchText$
`/`$SearchRegExp$
`/`$ReplaceRegExp$
`/`$ReplacementFlags$

$SearchText$ ::= $InterpolatedString$

$SearchRegExp$ ::= $InterpolatedString$

$ReplaceRegExp$ ::= $InterpolatedString$

$ReplacementFlags$ ::= { g |
$NumberOfReplacements$ }

3.3.5 Action Evaluation

Action Subspecifications in UGH come in two flavors: Assignments and Globalizations.

1. **Assignment Actions**
   Assignment Actions can be employed to assign a value to an UGH variable. They themselves can be further subcategorized into Clobbering Assignments, Nonclobbering Assignments, Global Assignments, and Regular Expression Assignments.

   (a) *Clobbering Assignments*
   Using a Clobbering Assignment will consign to the eternal void any value the variable to which a value is being assigned might previously have had. This is the "normal" assignment of such programming languages as C or Pascal.

   (b) *Nonclobbering Assignments*
   Using a Nonclobbering Assignment will assign a value to the variable named *VarName* if and only if *VarName* does not already have a value[7]

   (c) *Global Assignments*
   A GlobalAssignment is similar to a Clobbering Assignment, but the variable to which is assigned retains its value beyond the scope of the current Search Set. See Section 3.3.6 for details on UGH variables.

[7] if *VarName* is set to *VarNull*, a Nonclobbering Assignment will not clobber it, although in previous versions of UGH this was the case.

(d) *Regular Expression Transductions*
Using a Regular Expression Transduction assigns the string resulting from the replacement of the regular expression *SearchRegExp*, with *ReplaceRegExp* in the input string *SearchText* as a value to the variable named *VarName*. *ReplacementFlags* is either an integer indicating the number of replacements to perform or the letter 'g' for global[8] Although a full description of regular expressions is beyond the scope of this document[9], note that parenthesized subpatterns n ($1 \leq n \leq 9$) in *SearchRegExp* are available as $\backslash n$ in *ReplaceRegExp*.

2. **Globalization Actions**
Globalization Actions can be used to raise the scope of a variable from local Search Set scope to global scope. They come in both Clobbering and Nonclobbering varieties.

   (a) *Clobbering Globalizations*
   Like their Assignment counterparts, Clobbering Globalizations pay no heed to any previous value which may or may not be occupying the global variable named *VarName*.

   (b) *NonClobbering Globalizations*
   Nonclobbering Globalizations will only globalize the local value of *VarName* if at the time of evaluation *VarName* has no global value.

3.3.6 UGH Variables

Variables in UGH have much the same function as they do in other languages: as placeholders for values. Their values may be referenced in any *InterpolatedString*. Certain variables have special functions in UGH – notably *DefaultValueReference*.

When the value of a variable is referenced, it may be taken from one of

[8]The integer 0 also indicates a global replace.
[9]see `regexp.doc` or the `egrep`(1) manpage for more complete descriptions

three virtual namespaces within the UGH Engine, which are listed below in the order in which UGH traverses them.

1. **Search Set Local Variables**
   Variables assigned to by Clobbering, Nonclobbering, or Regular Expression Assignments are stored in a namespace local to the Search Set in which they occur[10].

2. **Search Set Criterium Values**
   The label=value pairs of a Search Set can themselves be referenced by means of the standard UGH variable notation. It is also possible to assign to a variable which carries a name identical to the label of some criterium present in a Search Set, although this has no effect whatsoever on the criterium value as interpreted by the UGH Engine.[11]

3. **Global Variables**
   An additional namespace for Global variables allows variable scope to extend to multiple Search Sets. Note that if a possible value within Search Set local scope exists, that value will be preferred to a global one. Currently there is no way to force a variable reference to be global, rather than local.

[10]This is not strictly true when recursive Search Sets are considered: more accurately, such variables are stored in a namespace local to the *top-level* Search Set in which they occur – this is both a Problem Of Recursive Search Sets and a Feature of UGH.

[11]This may change in the future, if the temporal ambiguities can be resolved – a more elegant solution would be a working implementation of recursive Search Sets, of course ...

3.4 UGH String Interpolation

3.4.1 Interpolated String Syntax

$$\begin{array}{lll} \mathit{InterpolatedString} & ::= \{ & \mathit{LiteralString} \mid \\ & & \mathit{VariableReference} \mid \\ & & \mathit{DelegateExpression} \mid \\ & & \mathit{UnescapeExpression} \mid \\ & & \mathit{DefaultValueReference} \;\}* \end{array}$$

$$\mathit{VariableReference} \;::=\; \mathit{ReferencePrefix}\texttt{\{}\mathit{VarName}\texttt{\}}$$ [12]

$$\mathit{DelegateExpression} \;::=\; \mathit{ReferencePrefix}\texttt{<}\mathit{MetaName}\ \mathit{MetaParam}\texttt{>}$$

$$\mathit{UnescapeExpression} \;::=\; \mathit{ReferencePrefix}\texttt{[}\mathit{InterpolatedString}\texttt{]}$$ [13]

$$\begin{array}{lll} \mathit{DefaultValueReference} & ::= \{ & \mathit{EscapedValueReference} \mid \\ & & \mathit{LiteralValueReference} \;\} \end{array}$$

$$\mathit{EscapedValueReference} \;::=\; \mathit{EscapedValuePrefix}\ \mathit{Underscore}$$

$$\mathit{LiteralValueReference} \;::=\; \mathit{ImmediateReferencePrefix}\ \mathit{Underscore}$$

$$\mathit{VarName} \;::=\; \mathit{InterpolatedString}$$

$$\begin{array}{lll} \mathit{ReferencePrefix} & ::= \{ & \mathit{ImmediateReferencePrefix} \mid \\ & & \mathit{DelayedReferencePrefix} \;\} \end{array}$$

$$\mathit{MetaName} \;::=\; \mathit{InterpolatedString}$$

$$\mathit{MetaParam} \;::=\; \mathit{InterpolatedString}$$

$$\mathit{Underscore} \;::=\; \texttt{\_}$$

[12] The curly brackets in *VariableReference* must appear as literals in the string to be interpolated and should not be confused with the curly brackets of the syntax notation.

[13] The square brackets in *UnescapeExpression* must appear as literals in the string to be interpolated and should not be confused with the square brackets of the syntax notation.

$EscapedValuePrefix$ ::= `@`

$ImmediateReferencePrefix$ ::= `$`

$DelayedReferencePrefix$ ::= `%`

3.4.2 String Interpolation Elaborations

"String Interpolation" refers to the process of substituting actual values for value references in some string occurring in an UGH specification. Details on the various types of value references in Interpolated Strings can be found in Section 3.4.3. Details on the actual temporal processes involved can be found in Section 3.4.4.

3.4.3 Value Reference Types

Interpolated Strings may contain literal substrings, interspersed with references to the following types of values:

1. **Variable Values**
   The value of an UGH variable will be substituted for a *VariableReference* within an Interpolated String at interpolation time[14], providing that the variable named *VarName* actually has a value in some accessible namespace[15] at this time. If the value of *VarName* is *VarNull*, then the empty string will be substituted for *VariableReference* in string into which values are being interpolated. Note that *VarName* is itself an Interpolated String, and therefore may itself contain value references[16]. Further note that the values of Search Set Criteria may themselves be referenced in this manner.

[14]see Section 3.4.4 for details on various interpolation-time processes.

[15]See Section 3.3.6 for details on UGH Variables and their namespaces.

[16]Such constructions are similar to "pass by reference" constructions in other languages. I will refer to them in the future as "higher order" value references. Given UGH's lack of temporal transparency, interpolation of higher order value references can become somewhat hairy.

2. **Meta Return Values**
The return value[17] of an UGH Meta Specification as invoked with *MetaParam* as its Parameter Value will be substituted for a *DelegateExpression* within an Interpolated String. The same rules concerning empty strings, *VarNull*, and higher order value references apply to *DelegateExpressions* as to *VariableReferences*. Higher order *DelegateExpressions* are especially troublesome entities, and Metas which are to be invoked in such a manner must be carefully constructed to deal with the possibility of receiving an only partially constructed Parameter Value – returning the empty string if this is the case.

3. **Unescaped Values**
Any character preceeded by a backslash (\) is considered to be "escaped", and appears as a literal (without the backslash) in any SQL query involving the string in which the escaped character occurs. Escaped characters are also treated as literals in RegExp search expressions. Characters which have special meanings for the UGH Engine are ignored during string interpolation if they are escaped – such characters include the following:

   `$ @ [ ] { } < > _ %` < *space* >

   An occurrence of an *UnescapeExpression* within an Interpolated String is replaced by an unescaped version of the *InterpolatedString* which is embedded within the *UnescapeExpression*. Backslashes functioning as escape prefixes are removed by this process.[18]

4. **Default Parameter Values**
The value of the Parameter[19] belonging to the UGH Specification in which the Interpolated String occurs is substituted for any occurrence of *DefaultValueReference* within such a string The literal Parameter Value is substituted for occurrences of *LiteralValueReference*, whereas

[17] See Section 3.3 for details on UGH Meta Specifications.

[18] This allows yet another method of simulating pass-by-reference within the String Interpolation paradigm, but has its own quirks.

[19] See the Sections concerning the individual UGH Specifications for details on how their Parameter values are defined.

the value substituted for *EscapedValueReference* is the result of auto-escaping[20] the literal Parameter Value. In the future, the assertion that a string S is "interpolated with resect to" some value Y should be taken to mean that the value Y will be taken as the Parameter Value for the purposes of the Interpolation.

3.4.4 String Interpolation Processes

The process of string interpolation can be broken down into the following stages:

1. **Default Interpolation**
   During the Default Interpolation stage, the interpolation of default parameter values as described in Section 3.4.3 is performed.

2. **Named Interpolation**
   During the Named Interpolation stage, the interpolation of variable values, meta return values, and unescape expressions is performed.

3. **Delayed Interpolation**
   During the Delayed Interpolation stage, both Default Interpolation and Named Interpolation are performed on value references which have *DelayedReferencePrefix* as their prefix.

Various stages of string interpolation itself is performed at the following times during the operation of the UGH Engine:

1. When generating an embedded Search Set by means of a Call Subspecification in some Context Specification, both the Name and Value parts of the Call Subspecification are interpolated[21] before they are placed into the embedded Search Set.

[20] **Auto-escaping** is a process by which characters in a string which may have special meaning for the UGH Engine itself are automatically prefixed by a backslash.

[21] In the normal sense of "interpolated," Default interpolation is performed before Named interpolation, and Delayed interpolation is not performed.

2. Upon encountering a non-ignored value for the name of some Extended Specification (either a Projection or Selection Specification) in some Search Set, the value of that Extended Specification's Table Subspecification is interpolated before it is added to the list of queried Relations for that Search Set.

3. When generating an SQL Projection String by means of an As Subspecification in some non-ignored Projection Specification, that As Subspecification is interpolated.

4. When generating an SQL Selection String by means of a Query Subspecification in some non-ignored Selection Specification, that Query Subspecification is interpolated.

5. When attempting to match a Search Set value to a Regular Expression Case Subspecification of some Selection Specification, both the Search and Replace parts of the Regular Expression Case Subspecification are interpolated.

6. When attempting to match a Search Set value to a Literal Case Subspecification of some Selection Specification, the value of the Literal Case Subspecification is interpolated.

7. During the evaluation of some Assignment Action occurring during the execution of some Meta, if the assignment takes place, then Default Interpolation is performed on the value expression occupying the right-hand side of the Assignment Expression.

8. During the substitution phase[22] of generation, full interpolation[23] is performed on any SQL Selection Conditions generated by any Query Subspecification during processing of the immediately preceeding set of search criteria.

   NOTE: This differs significantly from previous versions of UGH, in which the substitution phase interpolated strings from *all* preceeding search sets.

[22]Following both the primary context matching process itself and the execution of Metas whose On Subspecification is equal to *SearchSetEnd*

[23]Default Interpolation, Named Interpolation, and Delayed Interpolation

9. During the substitution-phase of generation, full interpolation is performed on any SQL Projection Conditions generated by any As Subspecification during the immediately preceeding search set.

   See **NOTE** above.

10. During the substitution-phase of generation full interpolation is performed on all queried Relations generated by an In Subspecification of any non-ignored Extended Specification during the immediately preceeding search set.

    See **NOTE** above.

11. When Interpolating some higher order value reference within an interpolated string, that higher order value reference is interpolated.

3.5 UGH Table Specifications

3.5.1 Table Syntax

$$\begin{array}{lll} Table-Specification & ::= & \texttt{TABLE:}TableName \; <nl> \\ & & [\texttt{JOINKEYS:}[JoinKeyList]] \; <nl> \end{array}$$

$$TableName \;\; ::= \;\; LiteralString$$

$$JoinKeyList \;\; ::= \;\; JoinKey[\texttt{,}JoinKeyList]$$

$$JoinKey \;\; ::= \;\; JoinKeyHandle[\texttt{=}JoinKeySQL]$$

$$JoinKeyHandle \;\; ::= \;\; LiteralString$$

$$JoinKeySQL \;\; ::= \;\; SQLColumnName \mid VirtualJoin$$

$$SQLColumnName \;\; ::= \;\; LiteralString$$

$$VirtualJoin \;\; ::= \;\; \texttt{<}JoinPath$$

$$JoinPath \;\; ::= \;\; ExternalTableName \; [VirtualJoin]$$

$ExternalTableName \quad ::= \quad LiteralString$

3.5.2 Notes on the Table Syntax

An UGH Table Specification serves to define a relation which may occur in an In Subspecification of some Extended Specification. Join Keys are declared in association with Tables to allow UGH to automatically construct inner join clauses during SQL generation.

3.5.3 Join Construction

UGH attempts to find the "best" join possible for each of the tables in the set of queried Relations for an UGH-generated SQL query. Given a set of n tables, UGH attempts to generate exactly $n - 1$ Join Clauses, but will add external tables to the set of queried Relations if necessary, adding additional Join Clauses to compensate.[24] If UGH receives a set of queried Relations which cannot be internally joined by any means at UGH's disposal, an exception (`EUGHCannotJoin`) will be raised.

The order of the keys in the JoinKeys Subspecification is considered to be a preference-order for the construction of joins: the first JoinKey is assumed to be the most specifc, and so on.

UGH will add a join clause of the form $T_1.j_1 = T_2.j_2$ to an SQL query during generation if the following conditions apply:

[24]It has come to my attention that the algorithm I have attempted to describe here (and to implement in UGH) is essentially a solution to the Steiner tree problem for graphs, in which the edge-weights correspond to the JoinKey hierarchies of individual tables, and the terminal nodes correspond to the list of queried Relations for a given search set. Since solutions to the Steiner tree problem are notoriously $\mathcal{NP}$ complex, I have opted to require the encoding of explicit *JoinPaths* in an UGH Database Specification. Any two tables, therefore, which may be queried together alone by a given search set, must be adjacent to one another by some join key, even if that join key is of the *VirtualJoin* type described above. Future implementations of UGH will quite likely address this problem in new and interesting (if not especially efficient) ways.

- Let T_1 be a Table with JoinKeys $j_{1.1}, \ldots, j_{1.n}$ and let T_2 be a Table with JoinKeys $j_{2.1}, \ldots, j_{2.m}$. Assume that T_1 and T_2 are both present in the list of queried Relations for a given Search Set.

- If there exist two JoinKeys $j_{1.i}$ and $j_{2.k}$ such that

  1. $1 \leq i \leq n$, and
  2. $1 \leq k \leq m$, and
  3. neither $j_{1.i}$ nor $j_{2.k}$ is equal to the empty string, and
  4. the *JoinKeyHandle* part of $j_{1.i}$ is literally equal to the *JoinKey-Handle* part of $j_{2.k}$, and
  5. there is no $i' < i$ and no $k' < k$ such that the preceeding conditions apply to i' and k' in place of i and k, respectively,

  then

- if the *VirtualJoin* part of $j_{1.i}$ is present, then all *ExternalTableName* parts of that *VirtualJoin* are added to the working set of queried Relations, and the attempt to construct a Join Clause for T_1 is restarted,[25]

- if the *SQLColumnName* part of $j_{1.i}$ is present and nonempty, then j_1 is equal to the *JoinKeySQL* part of $j_{1.i}$, otherwise j_1 is equal to the *JoinKeyHandle* part of $j_{1.i}$, and

- if the *SQLColumnName* part of $j_{2.k}$ is present and nonempty, and does not have the form of a *VirtualJoin*, then j_2 is equal to the *JoinKeySQL* part of $j_{2.k}$, otherwise j_2 is equal to the *JoinKeyHandle* part of $j_{2.k}$.

[25] assumedly, a direct join will be available for any tblae T_1 which references some table T_3 as the *ExternalTableName* part of a *VirtualJoin* for some *JoinHandle*

3.6 UGH Selection Specifications

3.6.1 Selection Specification Syntax

$$
\begin{aligned}
Selection-Specification \ ::= \ & \texttt{KEY:}KeyName \ < nl > \\
& [\texttt{IGNORE:}IgnoreVal \ < nl >] \\
& [\texttt{DISMISS:}DismissVal \ < nl >] \\
& [\texttt{IN:}[TableList] \ < nl >] \\
& [\texttt{PRIORITY:}PriorityString \ < nl >] \\
& [\texttt{CASE:}Case-Definition > < nl >]* \\
& [\texttt{PRE:}InterpolatedString \ < nl >] \\
& [\texttt{QUERY:}InterpolatedString \ < nl >] \\
& [\texttt{POST:}InterpolatedString \ < nl >] \\
& [\texttt{EXEC:}ExecMetaName \ < nl >] \\
& [\texttt{CALL:}SubContext \ < nl >]*
\end{aligned}
$$

$$TableList \ ::= \ TableName[\texttt{,}TableList]$$

$$TableName \ ::= \ InterpolatedString$$

$$PriorityString \ ::= \ InterpolatedString$$

3.6.2 Notes on the Selection Specification Syntax

At match time, if a criterium is found in the Search Set with the form *CriteriumName=CriteriumValues* and if *CriteriumName* is literally equal to *KeyName*, and if the string *CriteriumValues* is nonignored with respect to the Selection Specification, then an SQL Selection Condition is added to the query being constructed for that Search Set, later to emerge in the `WHERE` clause of the final SQL query. Given the follwing,

$$CriteriumValues ::= Subvalue_1 \ Connective-Char_1 \ \ldots \ Subvalue_n$$

the Selection Condition to be added has this form:

$$
\begin{array}{l}
(\ \ Pre-Query \\
\quad (\ \ Query-Condition_1 \quad Logical-Connective_1 \\
\quad\quad Query-Condition_2 \quad Logical-Connective_2 \\
\quad\quad \vdots \\
\quad\quad Query-Condition_n\ \ ) \\
\quad Post-Query\ \ )
\end{array}
$$

here, Query-Condition$_i$ $(1 <= i <= n)$ represents the result of interpolating[26] an individual subvalue[27] *Subvalue$_i$* of *CriteriumValues* into the *InterpolatedString* given in the Key Specification's Query Subspecification. Subvalues are separated by *Connective-Char* elements: characters representing the logical connectives conjunction and disjunction. By default, the UGH Engine recognizes the following logical separators in such *CriteriumValues*:

$$Connective-Char \quad ::= \quad And-Char \mid Or-Char$$

$$And-Char \quad ::= \quad \texttt{\&}$$

$$Or-Char \quad ::= \quad ;$$

In constructing a Selection Condition for some Selection Specification and some nonignored *CriteriumValues* as described above, the UGH Engine iterates over any and all subvalues *Subvalue$_i$* of *CriteriumValues*, assigning each one in turn to the Parameter Value of the Selection Specification, and adding the string resulting from the interpolation of the Query Subspecification of that Selection Specification to the Engine's Selection Conditions for that Search Set. The SQL logical connectives 'AND' and 'OR' occur to separate these individual Selection Conditions in positions corresponding to the occurrences of the logical separator characters in the *CriteriumValues* string.

NOTE: In older versions of UGH, round parentheses occurring in the subvalue-intial or in subvalue-final positions in a *CriteriumValues* string were

[26]See Section 3.4 for details on interpolated strings

[27]More accurately, the Key Specification's Parameter Value with respect to that subvalue: see Section 3.7 for details on how these may differ.

treated specially. This behavior has since been eliminated, since it did not deal at all well with subvalues which actually did contain parentheses, and since no one seemed interested in using the feature, anyways. Future versions of UGH may re-implement this feature, solving the problem of parenthesis-containing values either by a simple escape-check mechanism, or (preferably) by more sohpisticated parsing of logical expressions themselves.

An arbitrary Meta Specification[28] may be called by name from within a matched and nonignored Selection Specification by means of an Exec Subspecification. A Meta Specification invoked in this way inherits the current Parameter Value of its caller.

Since one may not wish a Meta Specification named in a Selection Specification's Exec Subspecification to be called for every subvalue of *CriteriumValues*, the matching process keeps track of the logical connective characters already parsed, and ceases to call such Meta Specifications after encountering an *Or-Char*.

To allow for discrepancies between the treatment of logical connectives in the query and the execution of referenced Meta Specifications, the following special sequences are accepted in *CriteriumValues*, where v_1 and v_2 represent the individual subvalues to be interpolated into the Key Specification's Query Subspecification, and where the column "Exec v_2" indicates whether or not a Meta Specification named in the Key Specification's Exec Subspecification will be called for the subvalue v_2.

Sequence	**SQL**	**EXEC v_2**
v_1 *And-Char* v_2	v_1 AND v_2	yes
v_1 *Or-Char* v_2	v_1 OR v_2	no
v_1 *And-Char Or-Char* v_2	v_1 AND v_2	no
v_1 *Or-Char And-Char* v_2	v_1 OR v_2	yes

After all matching processes have taken place, and before the construction of the SQL WHERE clause, the Selection Conditions for each Search Set are sorted according to their Priority Values. The value of interpolating a Selection Specification's Priority Subspecification, if present, should yield a

[28]See Section 3.3 for details on Meta Specifications.

string representing a valid decimal integer. If no Priority Subspecification is present, or if interpolating a Priority Subspecification does not yield such a string, then the Selection Condition generated is assigned the Priority *UGHDefaultPriority* (usually zero).

Selection Conditions for a given Search Set are sorted in descending order of priority. If multiple Search Sets are present, then the priority values of the Selection Conditions each contains are averaged, and the sets of Selection Conditions are themselves sorted according to the resulting value.

3.7 The Case Macro-Definition Mechanism

3.7.1 Case Syntax

$$Case - Definition \;::=\; \{ \; CaseLiteralClause \mid CaseRegExpClause \mid CaseDefaultClause \; \}$$

$$CaseLiteralClause \;::=\; CaseFrom[\texttt{=}CaseTo]$$

$$CaseRegExpClause \;::=\; RegExpFrom \; RegExpAssign \; RegExpTo$$

$$RegExpAssign \;::=\; \texttt{\~{}=}$$

$$RegExpFrom \;::=\; SearchRegExp$$

$$RegExpTo \;::=\; ReplaceRegExp\texttt{/}ReplacementFlags$$

$$CaseDefaultClause \;::=\; DefaultValueReference[= CaseTo]$$

$$CaseFrom \;::=\; LiteralString$$

$$CaseTo \;::=\; InterpolatedString$$

3.7.2 Notes on the Case Syntax

The Parameter Value[29] of a Selection Specification is determined with respect to an individual subvalue $Subvalue_k$ of a nonignored *CriteriumValues* string occurring in some search set,[30] and with respect to any Case Subspecifications Case Subspecifications $Case_1$,...,$Case_m$ contained in the Selection Specification.

- If the Key Specification has no nonempty Case Subspecifications, ($m = 0$), then the Parameter Value is $Subvalue_k$ itself.
- Otherwise, the Parameter Value is the Case Value of the first[31] Case Subspecification $Case_i$ which matches $Subvalue_k$ according to the rules described in Section 3.7.3, or $Subvalue_k$ itself, if no such matching $Case_i$ is found.

3.7.3 Case Match Types

- **Literal Match**
  A *CaseLiteralClause* of the form *CaseFrom=CaseTo* is said to match a subvalue $Subvalue_k$ if its *CaseFrom* part is literally equal to $Subvalue_k$.

  The **Case Value** of such a match is the result of interpolating *CaseTo* with respect to the default value $Subvalue_k$.
- **Regular Expression Match**
  A *CaseRegExpClause* is said to match a subvalue $Subvalue_k$ if that $Subvalue_k$ matches the regular expression resulting from interpolating its *SearchRegExp* part with respect to the default value $Subvalue_k$.

  The **Case Value** of such a match is the result of performing the replacement indicated by interpolating *ReplaceRegExp* with respect to

[29] The Parameter Value is that to which *DefaultValueReference* refers.

[30] See Section 3.6 for details on the syntax and processing of Selection Specifications.

[31] Strictly speaking, there may be no $i' < i$ such that $Case_{i'}$ matches $Subvalue_k$ according to the rules described in Section 3.7.3.

the default value *SubValue*$_k$.[32]

- **Default Match**
  A *CaseDefaultClause* will match any subvalue *Subvalue*$_k$.

  The **Case Value** of such a match is the result of interpolating *CaseTo* with respect to the default value *Subvalue*$_k$.

3.8 UGH Projection Specifications

3.8.1 Projection Specification Syntax

$$
\begin{aligned}
Projection - Specification \quad ::= \quad & \texttt{COL:}ColName \;\; < nl > \\
& [\texttt{IGNORE:} < IgnoreValue > < nl >] \\
& [\texttt{DISMISS:}DismissVal \;\; < nl >] \\
& [\texttt{IN:}[TableList] \;\; < nl >] \\
& [\texttt{PRIORITY:}PriorityString \;\; < nl >] \\
& [\texttt{AS:}ProjectionReference \;\; < nl >] \\
& [\texttt{EXEC:}ExecMetaName \;\; < nl >] \\
& [\texttt{CALL:}SubContext \;\; < nl >]*
\end{aligned}
$$

$$ProjectionReference \quad ::= \quad ProjectionItem[, ProjectionReference]$$

$$ProjectionItem \quad ::= \quad InterpolatedString\ [\ \texttt{AS}\ InterpolatedString]$$

3.8.2 Notes on the Projection Specification Syntax

At match time, if a criterium is found in some Search Set with the form *CriteriumLabel=CriteriumValue* and if *CriteriumLabel* is literally equal to *ColName*, and if *CriteriumValue* is nonignored with respect to the Projection Specification namded *ColName*, then a Projection Reference is added to the

[32] *ReplaceRegExp* is interpolated before the regular expression replacement is performed – this can cause problems, for example, if you want to pass a subpattern as Parameter to some Meta. You might consider using *DelayedReferencePrefix* in front of the *DelegateExpression* in such a situation.

query being constructed for that Projection Specification, later to emerge in the `FROM` clause, of the final SQL query, and possibly in the `ORDER BY` clause, as well.

The Projection Reference to be added is a list of all unique *Projection-Items* in the Projection Specification's As Subspecification. The uniqueness condition on Projection Items is automatically imposed by UGH on a query-wide basis.

After all matching processes have taken place, and before the construction of the SQL SELECT and/or ORDER BY clauses, the Projection References for the given Search Set(s) are sorted according to their Priority Values. The value of interpolating a Projection Specification's Priority Subspecification, if present, should yield a string representing a valid decimal integer. If no Priority Subspecification is present, or if interpolating a Priority Subspecification does not yield such a string, then the Projection Reference generated is assigned the Priority *UGHDefaultPriority* (usually zero).

Projection References are sorted in descending order of priority before being integrated in any overt SQL clauses. If the Priority Value carried by a Projection Reference is less than or equal to *UGHDefaultPriority*, then nothing is added to the ORDER BY clause based on that Projection Reference.

Chapter 4

UGH Database Conventions

4.1 Notes on the Database Notation

In the following sections, table structures for an UGH source database will be described. The following conventions for the description of these structures will be used:

*	indicates the Primary Key
+	indicates a required (NOT NULL) field

In the textual elaborations following the table-structure schematics, the following conventions will be employed:

- **Boldface Text** will be used to refer to the names of tables and the names of columns.
- *Slanted Text*, written as the name of some column, will be used to refer to an abstract value of that column.
- `Fixed-Width Text` will be used to designate literal values in the database, and also for example queries
- References to equivalent textual representations will make use of the

text syntax notational conventions as described in Section 3.1.

4.2 Notes on the Database Content

4.2.1 Database Structure

An UGH Source Database has the general structure schematically presented in Figure 4.1. This schematic was last updated for UGH Version 1.6.4.

In an UGH Source Database, the information which defines UGH Specifications is organized according to the following hierarchy:

- **Table Specifications**
  Primary Table(s): Tables
  Referencing Table(s): -
  Supporting Table(s): JoinKeys
  Type Identifier: 2
- **Context Specifications**
  Primary Table(s): Contexts
  Referencing Table(s): -
  Supporting Table(s): SubContexts, Metas, Extendeds
  Type Identifier: 1
  - **SubContexts Subspecifictions**
    Primary Table(s): SubContexts
    Referencing Table(s): Contexts
    Supporting Table(s): -
    Type Identifier: -
  - **Meta Specifictions**
    Primary Table(s): Metas
    Referencing Table(s): Contexts
    Supporting Table(s): Actions
    Type Identifier: 1

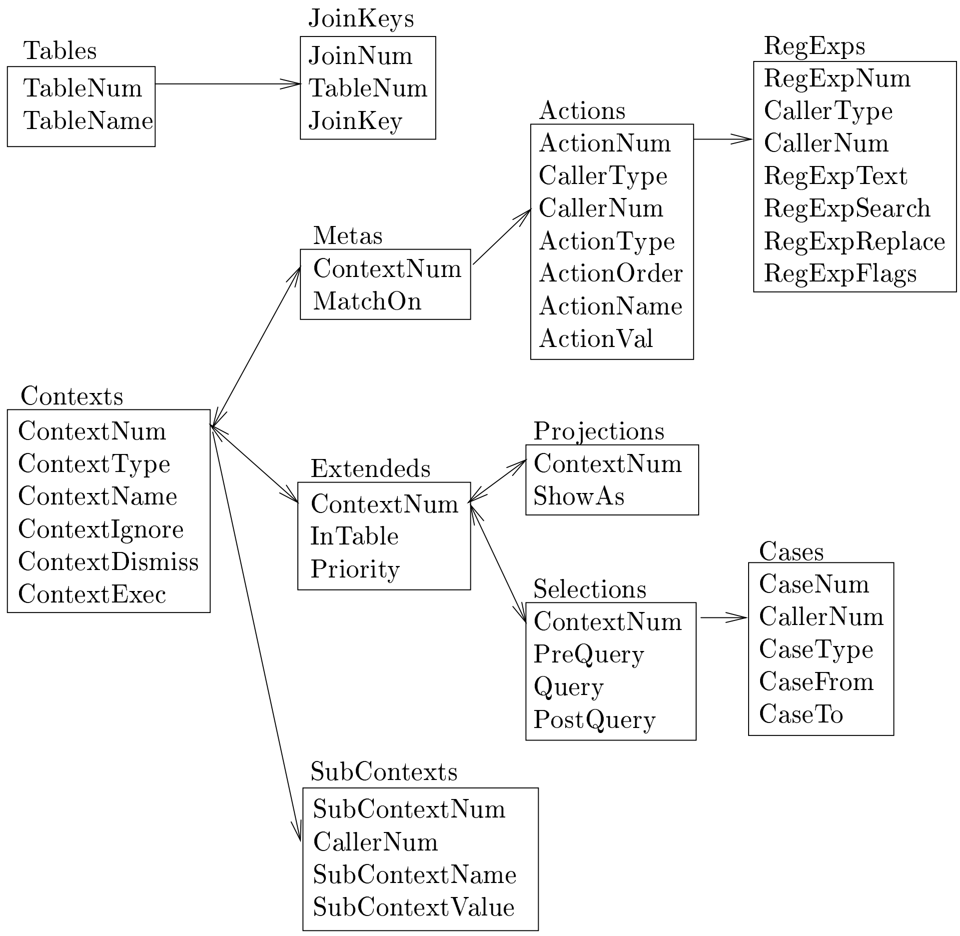

Figure 4.1: UGH Source Database Structure

* **Action Subspecifications**

  Primary Table(s): Actions
  Referencing Table(s): Metas
  Supporting Table(s): RegExps
  Type Identifier: 3

- **Extended Specifications**

  Primary Table(s): Extendeds
  Referencing Table(s): Contexts
  Supporting Table(s): Selections, Projections
  Type Identifier: -

  * **Projection Specifications**

    Primary Table(s): Projections
    Referencing Table(s): Extendeds
    Supporting Table(s): -
    Type Identifier: 3

  * **Selection Specifications**

    Primary Table(s): Selections
    Referencing Table(s): Extendeds
    Supporting Table(s): Cases
    Type Identifier: 2

    · **Case Subspecifications**

      Primary Table(s): Selections
      Referencing Table(s): Cases
      Supporting Table(s): -
      Type Identifier: 4

4.3 The "Tables" Table

Table Name: Tables	
Column Name	**Column Type**
* TableNum	int
+ TableName	char(255)

The **Tables** Table, together with the **JoinKeys** Table, serves to define UGH Table Specifications, as does a *Table-Specification* Declaration as described in Section 3.5.

4.4 The "JoinKeys" Table

Table Name: JoinKeys	
Column Name	**Column Type**
* JoinNum	int
+ TableNum	int
+ JoinKey	char(255)

The **JoinKeys** Table performs the function of a JoinKey Subspecification of an UGH Table Specification as described by *JoinKeyList* in Section 3.5. For each *JoinKey* in the *JoinKeyList* of a textual source, one record should appear in the **JoinKeys** Table of a database source.

4.5 The "Contexts" Table

Table Name: Contexts	
Column Name	**Column Type**
* ContextNum	int
+ ContextType	int
ContextName	char(255)
ContextIgnore	char(255)
ContextDismiss	char(255)
ContextExec	char(255)

The "Contexts" Table contains the primary key for most UGH Specifications, namely *ContextNum*. This table contains the information common to the different types of UGH Context Specifications, which include:

- Meta Specifications, which are further defined by the **Metas**, **Actions**, and **RegExps** tables;
- Extended Specifications, which are further defined by the **Extendeds** table and include:
  - Selection Specifications, which are themselves further defined by the **Selections** and **Cases** tables; and
  - Projection Specifications, which are further defined by the **Projections** table.

More information common to all Context Specifications can be found in the **SubContexts** Table.

Currently implemented values for *ContextType* are the following:

Subtype Name	ContextType
Unknown	0
Meta	1
Selection	2
Projection	3

4.6 The "SubContexts" Table

Table Name: SubContexts		
Column Name		Column Type
*	SubContextNum	int
+	CallerNum	int
+	SubContextName	char(255)
	SubContextValue	char(255)

The **SubContexts** Table serves to define Call Subspecifications for all UGH Context Specifications.

4.7 The "Extendeds" Table

Table Name: Extendeds		
Column Name		Column Type
*	ContextNum	int
	InTable	char(255)
	Priority	char(255)

The **Extendeds** Table serves to define that which is common to all UGH Extended Specifications, which include

- Selection Specifications, which are further defined by the **Selections** and **Cases** tables; and
- Projection Specifications, which are further defined by the **Projections** table.

4.8 The "Metas" Table

Table Name: Metas	
Column Name	**Column Type**
* ContextNum	int
MatchOn	char(255)

The **Metas** Table, together with the**Contexts** Table and the **Actions** Table serves to define UGH Meta Specifications, as does a *Meta-Specification* Declaration as described in Section 3.3.

4.9 The "Actions" Table

Table Name: Actions	
Column Name	**Column Type**
* ActionNum	int
+ CallerType	int
+ CallerNum	int
+ ActionType	int
+ ActionOrder	int
+ ActionName	char(255)
ActionVal	char(255)

The **Actions** Table serves the function of definining Action Subspecifications as described in Section 3.3.4.

The correspondence between the operators in a `DO:` clause in a textual source and *ActionType*[1] in a database source is as follows:

Subtype	**Operator**	**ActionType**
Unknown		0
CheckSet	`?=`	1
NoCheckSet	`:=`	2
GlobalSet	`!=`	3
CheckGlobalize	`?!`	4
NoCheckGlobalize	`:!`	5
RegExp	`~=`	6

CallerType corresponds to some Major (sub-)Specification Types:

Major Type	**Type Identifier**
Unknown	0
Context	1
Table	2
Action	3
Case	4

ActionName corresponds to the name of the variable whose value is to be set, globalized, or otherwise transformed by the Action Subspecification.

ActionVal is used only by Assignment Actions, and corresponds to the value to be assigned to the variable.

Further information regard Regular Expression Transformation Action Subspecifications is extracted from the **RegExps** Table.

[1]values in the **ActionType** column of the **Actions** table

4.10 The "RegExps" Table

Table Name: RegExps		
	Column Name	**Column Type**
*	RegExpNum	int
+	CallerType	int
+	CallerNum	int
+	RegExpText	char(255)
	RegExpSearch	char(255)
	RegExpReplace	char(255)
	RegExpFlags	int

The **RegExps** Table serves to define Regular Expression Transformation Assignment Action Subspecifications, as described in Section 3.3.5. Note that due to the fact that *RegExpFlags* must be an integer in a database source, the value "g" is no longer allowed – use 0 (zero) instead.

CallerType is a major type – currently, only Action Subspecifications reference the **RegExps** Table, therefore the only value that ought to be found in *CallerType* is 3.

4.11 The "Selections" Table

Table Name: Selections	
Column Name	**Column Type**
* ContextNum	int
PreQuery	char(255)
Query	char(255)
PostQuery	char(255)

The **Selections** Table, together with the **Extendeds**, **Contexts**, and **Cases** tables, serves the function of defining UGH Selection Specifications, as described in Section 3.6.

4.12 The "Cases" Table

Table Name: Cases	
Column Name	**Column Type**
* CaseNum	int
+ CallerNum	int
+ CaseType	int
CaseFrom	char(255)
CaseTo	char(255)

The **Cases** Table serves to define Case Subspecifications of Selection Specifications, as described in Section 3.7.2.

CallerNum is assumed to be a key to the **Contexts** Table, corresponding to some Selection Specification in the **Selections** Table.

The following values for *CaseType* are recognized:

Clause Type	**CaseType Identifier**
Unknown	0
Literal	1
Regular Expression	2

4.13 The "Projections" Table

Table Name: Projections	
Column Name	**Column Type**
* ContextNum	int
ShowAs	char(255)

The **Projections**, together with the **Extendeds**, and **Contexts** Tables, serves to define UGH Projection Specifications as described in Section 3.8.

Chapter 5

The UGH API

5.1 Notes on the API

UGH's Application Program Interface (API) is a Delphi package consisting of three components: TUGHEngine, TUGHSource, and TUGHDatabase. By default, these components can be found on the "Data Access" page of Delphi's component toolbar. A brief description of each UGH component follows.

TUGHEngine is the component primarily responsible for transforming a set of search criteria into an SQL query. Matching functions, interpolation functions, and data members representing the database specification itself, as well as the building blocks of an SQL query live here. Details concerning the TUGHEngine component may be found in Section 5.2, and in the file `UGHEngine.pas`.

A TUGHSource component may be used to handle the compilation of an UGH source file as described in Section 3 or of an UGH source database as described in Section 4, as well as to save an image of an existing TUGHEngine to either a file or a database. Details concerning the TUGHSource component may be found in Section 5.3, and in the file `UGHSources.pas`.

A TUGHDatabase component brings together a TUGHEngine component, a TUGHSource component, and a TQuery component. The TUGHDatabase component itself is derived from Delphi's TDatabase component, and in this guise represents the database to be queried by UGH. It is the most abstract and the most stable of the UGH components. Most user-level interaction with UGH should occur through a TUGHDatabase component. Details concerning the TUGHDatabase component may be found in Section 5.4, and in the file `UGHDatabase.pas`.

5.2 The TUGHEngine component

5.2.1 TUGHEngine: Manipulation Procedures

The following procedures may be used to manipulate an UGH database Specification. The individual specification types themselves are defined in the file `UnitUGHTypes.pas`.

```
Add(Spec: TDataSpec);
AddKey(KeySpec: TKeySpec);
AddTable(TableSpec: TTableSpec);
AddMeta(MetaSpec: TMetaSpec);
AddCol(ColSpec: TColSpec);
Clear;  { wipe everyting }
Reset;  { remove only match-sensitive information }
```

The “Reset” procedure is of particular interest – it should normally be called following the generation of each query.

5.2.2 TUGHEngine: Basic Generation Procedures

The following procedures may be used to generate an SQL query from given search criteria.

```
GenerateMulti(KeyValsList: TStringListList; SQL: TStrings);
GenerateSQL(KeyVals,SQL: TStrings);
```

Note that the "TStringListList" type required by "GenerateMulti" is declared in the file `UnitListClasses.pas`.

5.2.3 TUGHEngine: Modal Generation Procedures

The following procedures may be used (with care) to generate only parts of an SQL query, or to selectively reset the UGH Engine.

```
GenerateSelections(KeyValsList: TStringListList);
GenerateProjections(KeyValsList: TStringListList);
ClearSelections;
ClearProjections;
```

5.2.4 TUGHEngine: Filter Procedures

The following procedures may be used to destructively filter the search criteria they are given: ignored values are deleted from the given lists.

```
FilterMulti(KeyValsList: TStringListList);
FilterLocal(KeyVals: TStringList);
```

5.2.5 TUGHEngine: Interpolation Functions

The following functions may be used to access UGH's string interpolation mechanism.

```
GetVariableValue(VarName: string): string;
```

```
InterpolateDefaults(Str,Val: string): string;
{ InterpolateFull reads its DefaultVal from stack.peek! }
InterpolateFull(Str: string): string;
InterpolateString(Str: string; DefaultVal: string): string;
InterpolateDelayed(Str: string; DefaultVal: string): string;
```

5.2.6 TUGHEngine: Public Data Members

The following public data members may be set by the user to tune the UGH's behavior.

```
KeyValCommentChar: string;
{ debug variables}
HaltOnUnknownKey: boolean;
HaltOnUnjoinableTable: boolean;
OrderByDirection: string;
```

5.2.7 TUGHEngine: Published Properties

The following published properties may be read and manipulated to alter UGH's behavior, or to gather information concerning the internal state of the Engine.

```
{ --------------- Behavior Modifiers -------------- }
property FromTables: TUniqueStringSequence;
property GenerationMode: TUGHGenerationMode;
property QueryMode: TUGHQueryMode;

{ -------- Match-sensitive information ------------ }
property Vars: TUniqueStringList;
property Globals: TUniqueStringList;

{ ------- Database Specification (the guts) ------- }
```

```
property Keys: TUGHKeyList;
property Tables: TUGHTableList;
property Metas: TUGHMetaList;
property Cols: TUGHColList;
property Pass: integer;

{ -------------- SQL Building-blocks -------------- }
property SelectClause: TStringList;
property FromClause: TStringList;
property WhereClause: TStringList;
property OrderByClause: TStringList;
```

5.2.8 TUGHGenerationMode Constants

The following constants may be used as indices to set the flags in the GenerationMode property of a TUGHEngine component.

Symbolic Constant	Effect
ugmSelect	Generate adds only KEY: info to Engine
ugmProject	Generate adds only COL: info to Engine
ugmFilter	Generate destructively alters input S-Sets

5.2.9 TUGHQueryMode Constants

The following constants may be used as indices to set the flags in the QueryMode property of a TUGHEngine component.

Symbolic Constant	Effect
uqmSelect	Construct a vanilla SQL query
uqmDistinct	Construct a ”SELECT DISTINCT” query
uqmCount	Construct a ”SELECT COUNT()” query[1]
uqmFrom	Construct the ”FROM” Clause
uqmWhere	Construct the ”WHERE” Clause
uqmOrderBy	Construct the ”ORDER BY” Clause

5.2.10 TUGHEngine: Other Stuff

Other TUGHEngine internals are user-accessible, look in `UGHEngine.pas` for details, if you need them.

5.3 The TUGHSource component

5.3.1 TUGHSource: Compilation Functions

The following functions may be used to compile an UGH Specification. On successful compilation, these functions return “True”.

```
Compile(Engine: TUGHEngine): boolean;
CompileLocal(Engine: TUGHEngine; Specs: TStrings): boolean;
```

The “CompileLocal” function may be used to dynamically create an UGH Engine based on user input.

5.3.2 TUGHSource: Image Saving Procedures

The following procedures may be used to “uncompile” am existing TUGH-Engine component into either an UGH source file or an UGH source database.

[1]Currently unimplemented.

```
UnCompileText(Engine: TUGHEngine; SpecStrings: TStrings);
UnCompileDB(Engine: TUGHEngine; DestDB: TDatabase);
```

5.3.3 TUGHSource: Published Properties

The following properties may be read or set to alter the behavior of the "Compile" function.

```
property SourceType: TUGHSourceType;
property SourceName: string;
property SessionName: string;
```

Note that "SessionName" is only meaningful when "SourceType" = "ughSrcDB". Compilation behavior is defined as follows:

- If "SourceType" = "ughSrcDB", then the TDatabase named "SourceName", as retrieved from the TSession named "SessionName", is used as the source for compilation.
- If "SourceType" = "ughSrcFile", then the file named "SourceName" is used as the source for compilation.
- If the source cannot be determined, or if the indicated source cannot be found, compilation fails.

5.3.4 TUGHSourceType: Definition

The following enumerative type's constants may be used to set the "SourceType" property of a TUGHSource component.

```
type TUGHSourceType = (ughSrcNull,ughSrcDB,ughSrcFile);
```

5.4 The TUGHDatabase component

5.4.1 TUGHDatabase: Initialization Procedures

The following function may be used to initialize a TUGHDatabase component based on the contents of an ASCII source file. Note that this file is **NOT** an UGH Database Specification as described in Section 3, but rather a file which sets the properties of TUGHDatabase inherited from Delphi's own TDatabase component. Description of the syntax of the parameters file is beyond the scope of this document, but have a look at the source if you're interested.

```
LoadParameters(const ParametersFileName: string);
```

5.4.2 TUGHDatabase: Compilation Functions

The following functions may be used to compile the TUGHEngine component associated with the TUGHDatabase from the source specified by the TUGHSource component associated with the TUGHDatabase.

```
Compile: boolean;
CompileLocal(SpecStrings: TStrings): boolean;
```

5.4.3 TUGHDatabase: Annihilation Procedures

The following procedures may be used to consign information associated with a TUGHDatabase object to the great bit bucket in the sky.

```
Reset;
Clear;
```

Calling the "Clear" procedure removes all specifications from an associated TUGHEngine. The "Reset" procedure may be called to remove match-specific information subsequent to matching. In fact, the latter is recommended.

5.4.4 TUGHDatabase: Generation Procedures

The following procedures may be used to generate an SQL query from the given Search Set(s). Note that the TStringListList type is defined in the file `UnitListClasses.pas`.

```
GenerateQuery(KeyValList: TStringListList);
GenerateLocal(KeyValList: TStringListList; LocalSQL: TStrings);
GenerateQuerySimple(KeyVals: TStrings);
GenerateLocalSimple(KeyVals, LocalSQL: TStrings);
```

5.4.5 TUGHDatabase: Published Properties

The following properties of a TUGHDatabase component may be read or set.[2]

```
Compiled: boolean;
Query: TQuery;
Engine: TUGHEngine;
Source: TUGHSource;
```

Setting "Compiled" to "false" calls the "Clear" procedure. If a generation procedure is invoked while "Compiled" is set to "false", the "Compile" function is called automagically.

[2] Of course, the properities inherited from Delphi's TDatabase component may be read or set as well, but they are not my concern.

The “Query” property must be set (non-nil) if the “GenerateQuery” functions are to work properly.

The “Source” property must be set (non-nil) if the compilation functions are to work properly.

The “Engine” property must be set (non-nil) if just about anything is to work properly.