Q-Schema

Executive Overview

Q-Schema is a facility to describe objects which may be relatively simple, or massively complex. Basically, ***Q-Schema*** is a mechanism to describe to a software program how to obtain structured data either from an end user through data forms capture, or from a database storage facility after that data have been previously captured and stored. In addition to describing the objects themselves, ***Q-Schema*** has the ability to describe to any SQL based database management system how to store the objects, as well as their schema description within that database.

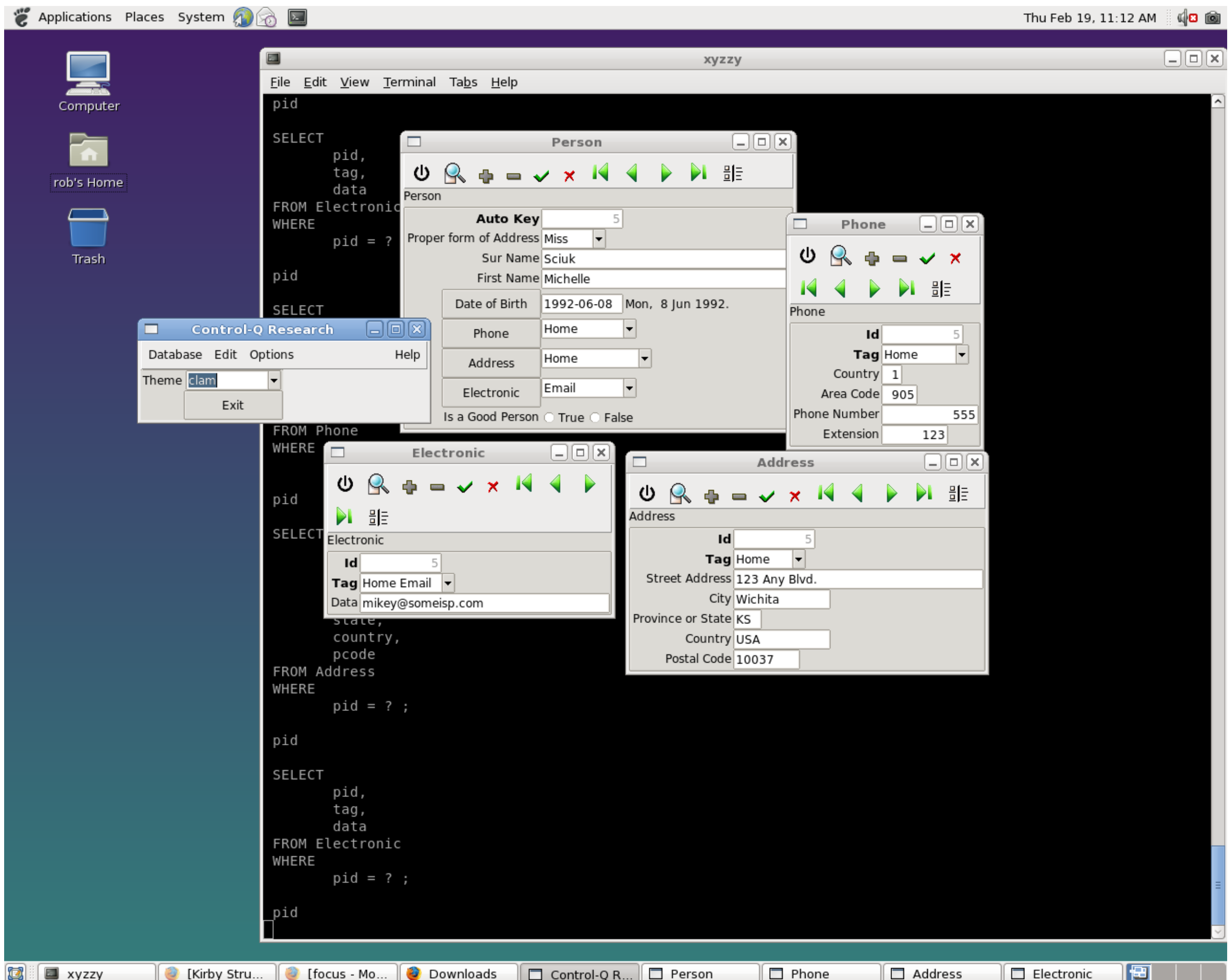
While on the face of it, this seems a simple concept, the idea of an active data dictionary being used in software tools is relatively rare. Most software is written against a specific schema, or data definition, and if that definition changes, then code changes are required throughout the application. The approach taken with ***Q-Schema*** is to have the data dictionary drive the application code, even to the point where code can be generated at runtime to manage the data objects. It's not magic, but it does lead to efficiencies on a scale which are unique to this approach.

Applications for the tool include commercial desktop applications tools with vertical application delivery, as well as a web based data source either in catalog form, or indeed with the full CRUD capability exposed to web based frameworks such as ***Dojo*** or ***jQuery.***

The Concept

A schema is comprised of Classes, which correspond closely to a database relation or table. The classes themselves have attributes, and in addition, may contain children which are Fields. As with the Schema and the Class, the Field maintains attributes which at a minimum include the tag or Field Name, a data type which corresponds closely to a common database type, but which has been extended to include common data types which are manipulated in the real world, but more on that later. By implementing this abstraction layer as fundamental to the Q-Schema facility, we have essentially provided a close mapping to a relational database, and thereby the means to define objects, to instantiate (or create an instance of) them, to store and retrieve them in practically any database, and to manipulate them in a trivial manner.

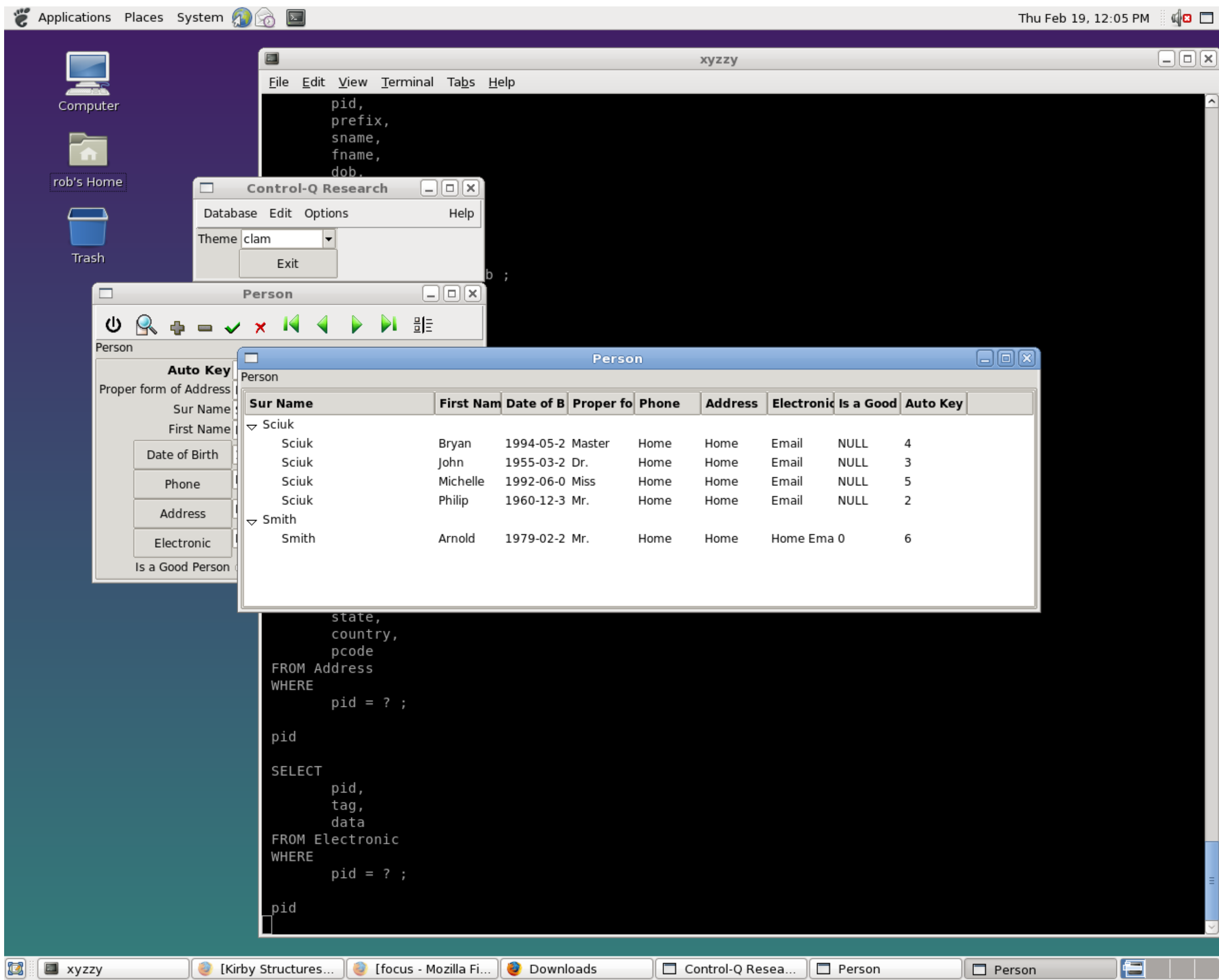
Ok, so much for the esoterica, what does this really mean? Let's demonstrate the concept with a simple example. We'll begin by defining objects which will correspond to real live things, and by things, in this case, I mean a person, and their contact information. Rather than type it all in, why not have Q-Schema just describe for us the schema which we've previously defined via the API (Application Programming Interface). See Appendix A for the listing. While this may be of interest to the more technically inclined, for the purpose of this paper, we will focus on the benefits of *having* an active data dictionary rather than the details of producing it. In keeping, the standard database ***CRUD*** (Create, Retrieve, Update, Delete) facility is only a single mouse-click away, and the following screens are generated automatically.



Note that rather than try to stuff multiple addresses, phone numbers and electronic contents into the Person class, each were given their own classes, and a link to those classes was provided in the Person class, via the data dictionary Lookup type. This leads to proper database normalization, and costs nothing to the developer, as Lookups are handled in an appropriate manner by the generated ***CRUD*** code. In the prototype below, note too that the forms displayed have the ability to change themes, and this currently runs on Unix, Linux, Mac/OSX and Windows platforms with a host consistent look and feel.

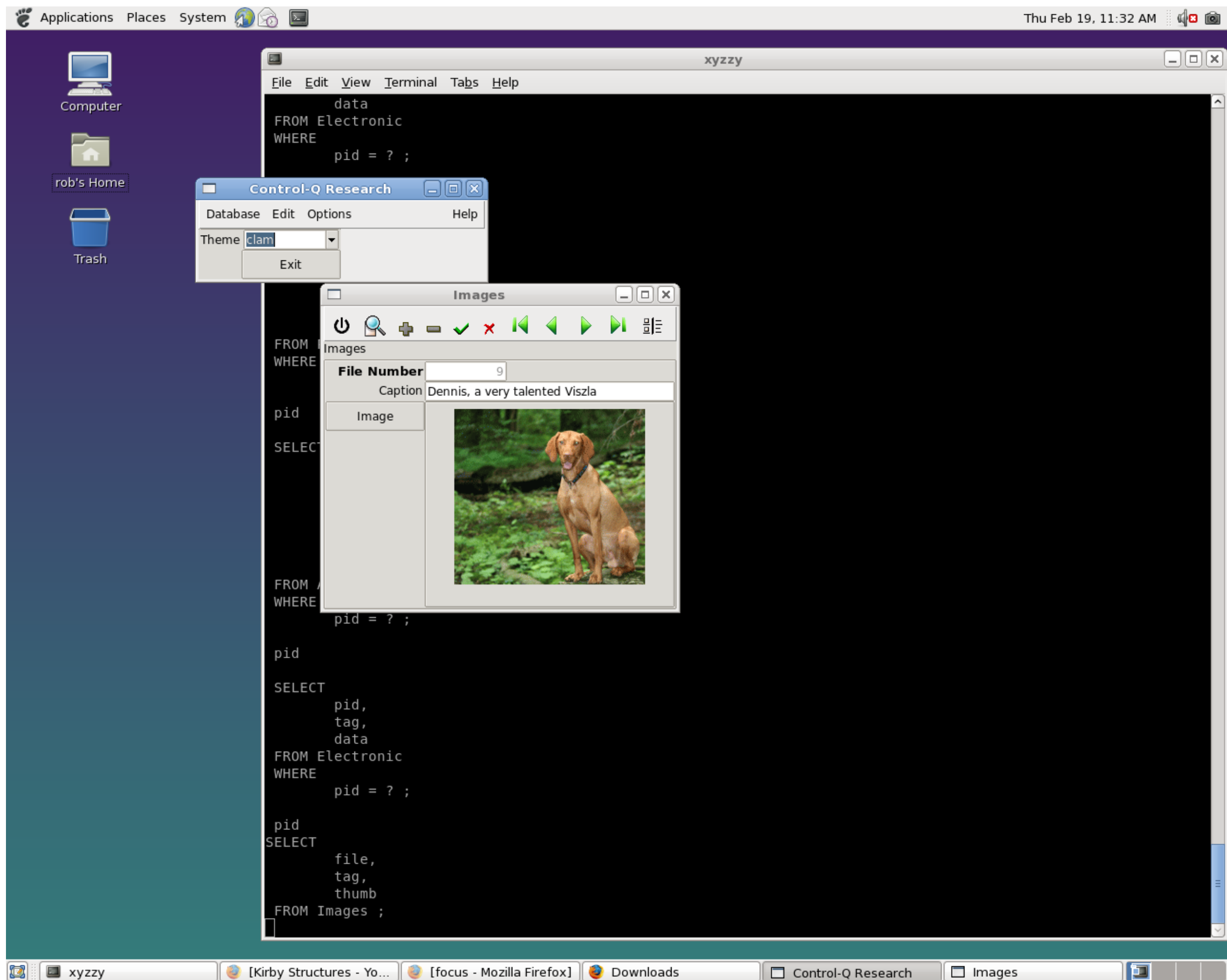
Another important aspect of ***Q-Schema*** for those who require a general purpose data dictionary facility, but need to have their own abstractions, the ability to define arbitrary objects (not simply Schema Class Field Attribute) also exists. This means that practically anything can be modeled, and the application developer needs only grab the attributes of interest, and utilize them in the manner which is suitable. This makes ***Q-Schema*** much more powerful, and sample applications will be forthcoming (watch this space).

Note that no code was required to generate these forms, and that the schema information was retained within the application which holds the data. In essence, ***Q-Schema*** describes each data object, the relationship between those objects, and the access method. Note that data retrieval are trivial, and a search can be conducted upon any field. Should the search return more than a single record, then the ability to pop up an en-tabulated form of the result set will allow the user the ability to drill down right to the required record in short order. Note that in classes where indices are present, as defined within the data dictionary, then the fields in the en-tabulated view will be re-ordered to reflect a natural organization, and in the case shown below, they are clustered by surname. This is the program default, and requires no additional thought by either the user, or the application designer.



Data Types

Q-Schema understands a number of data types, including the standard string, text, numeric and dates. In addition, various image data are supported via the integrated blob handling facility. This differs from other systems, in that blob data are handled in a manner completely consistent with other data, and are efficiently managed, and copied only when absolutely necessary. By linking with third-party graphics manipulation libraries, applications can be easily made which will re-size, convert image file formats, or perform other operations upon the data. In a similar manner, blob type data can be manipulated in accordance to data maintained within the dictionary. In the case of the Image class schema below, the geometry of the display image is specified within the schema data itself, and speaks directly to the embedded application generation tool.



The Programming API's

Q-Schema was written in a portable dialect of the ANSI-C language, and takes advantage of no specific machine facilities. For access to the operating system, file system, and network I/O facility of each ported host platform, an additional library which encapsulates the ***Apache Portable Runtime*** has been made available to take advantage of host facilities. This makes porting ***Q-Schema*** to new hosts a snap. It also makes embedding the ***Q-Schema*** facility into a scripting language (PHP, Perl, Tcl/Tk, Python, Ruby, Forth etc) very simple. Currently, the ***Tcl/Tk*** script language is used to generate the CRUD application owing to the speed, simplicity of the language, and the inherent portability of that remarkable scripting language. With the advent of the recent theming widgets, a modern look and feel is achieved on each platform.

Schema API's – The Data Definition

We plan to produce a graphical tool to generate the data dictionary with a drag and drop style interface which will allow even naive end users the ability to create applications simply and easily. For now, the API remains the only tool for creation of the data definition, and this is described in ***Appendix B***. As it is, one simply declares a new schema, adds attributes, and classes. For each class, simply add attributes and fields, and for each field, its attributes need to be defined. Keys are attributes of classes, as are indices, which lead naturally to field ordering and re-ordering in the tabular drill-downs mentioned above.

Database Access – The Repository

Q-Schema is database agnostic. In keeping drivers for ***SQLite*** and ***PostGreSQL*** have been implemented, with ***MySQL*** underway, and commercial databases will follow as market demands are made clear.

In accessing the database, a repository layer introduces an extremely simple model which directly implements the CRUD facility and has proven to be very effective. Data are managed and manipulated in constructs called tuples, which contain both the instance data and the class definition of the object.

Each schema has classes and in turn each class object defines its field objects, and one can traverse the model in both the parent and child direction. Iteration over the classes of a schema, or the fields of a class are inherent in the model, and provide a complete and simple means to access the data. The specific database access method is chosen at runtime, and the repository API has a mechanism to specify different database drivers at run time, on a connection basis. This of course opens the door for applications which span not only databases, but even database management systems. Row operations use and return tuples, and select style operations return tables of tuples, which can be iterated over, and nested to perform network or hierarchical access to what is otherwise a relational data organization.

Another critical aspect of the Repository layer is to store and retrieve the data dictionary objects within the database. This requires the ability to store the dictionary within the database and currently ***Q-Schema*** is using Protocol V101. This very simple design is able to describe all objects, and in addition, creates an authentication layer where needed (users, groups, security levels). Schema objects are stored and retrieved, the objects themselves are instantiated by the creation of the database tables using SQL

DDL definitions generated by the system, and finally the data objects themselves are added and manipulated via the repository layer.

The Work Ahead

At the time of this writing, the conceptual and working model for ***Q-Schema*** are materially complete. Additional modules required for commercial distribution include the following:

- **Schema Tool**
- **Additional Database Drivers**
- **Stand alone Server/Web Interface**

The web interface is under way, and much of the code is complete (check back shortly for an update to this document). Web 2.x design will be supported, and support for modern web development techniques and frameworks will ensure ready acceptance into the commercial web sphere. Commercial database driver development (***SQLServer, Oracle, Sybase, DB2...)*** will be deferred until a clear priority has been established, in essence by market demand.

Appendix A: Q-Schema dump of the test schema.


```
<tst_Schema.Person type='t_Class'>
 <attributes>
  <index>
   sname
   fname
   dob
  </index>
  <key>pid</key>
 </attributes>
 <tst_Schema.Person.pid type='t_Field'>
  <attributes>
   <label>Auto Key</label>
   <type>db_Intkey</type>
  </attributes>
 </tst_Schema.Person.pid>
 <tst_Schema.Person.prefix type='t_Field'>
```

```
<attributes>
  <label>Proper form of Address</label>
  <type>db_Enum</type>
  <value>
   Mr.
   Mrs.
   Ms.
   Dr.
   Master
   Miss
   Rev
   Rabbi
   Sheikh
  </value>
 </attributes>
</tst_Schema.Person.prefix>
<tst_Schema.Person.sname type='t_Field'>
 <attributes>
  <label>Sur Name</label>
  <length>32</length>
  <type>db_String</type>
 </attributes>
</tst_Schema.Person.sname>
<tst_Schema.Person.fname type='t_Field'>
 <attributes>
  <label>First Name</label>
  <length>32</length>
  <type>db_String</type>
 </attributes>
</tst_Schema.Person.fname>
<tst_Schema.Person.dob type='t_Field'>
```

```
 <attributes>
  <format>yyyy.mm.dd</format>
  <label>Date of Birth</label>
  <type>db_Julian</type>
 </attributes>
</tst_Schema.Person.dob>
<tst_Schema.Person.phone type='t_Field'>
 <attributes>
  <class>Phone</class>
  <label>Phone</label>
  <length>10</length>
  <toggle>tag</toggle>
  <type>db_Lookup</type>
  <using>pid</using>
 </attributes>
</tst_Schema.Person.phone>
<tst_Schema.Person.address type='t_Field'>
 <attributes>
  <class>Address</class>
  <label>Address</label>
  <length>10</length>
  <toggle>tag</toggle>
  <type>db_Lookup</type>
  <using>pid</using>
 </attributes>
</tst_Schema.Person.address>
<tst_Schema.Person.internet type='t_Field'>
 <attributes>
  <class>Electronic</class>
  <label>Electronic</label>
  <length>10</length>
```

```
    <toggle>tag</toggle>
    <type>db_Lookup</type>
    <using>pid</using>
  </attributes>
 </tst_Schema.Person.internet>
 <tst_Schema.Person.isgood type='t_Field'>
  <attributes>
    <label>Is a Good Person</label>
    <type>db_Boolean</type>
  </attributes>
 </tst_Schema.Person.isgood>
</tst_Schema.Person>
<tst_Schema.Address type='t_Class'>
 <attributes>
   <key>
    pid
    tag
   </key>
 </attributes>
 <tst_Schema.Address.pid type='t_Field'>
  <attributes>
    <label>Id</label>
    <type>db_Integer</type>
  </attributes>
 </tst_Schema.Address.pid>
 <tst_Schema.Address.tag type='t_Field'>
  <attributes>
    <label>Tag</label>
    <length>12</length>
    <type>db_Enum</type>
    <value>
```

```
   Home
   Work
   Other
   Ship-To
   Bill-To
  </value>
 </attributes>
</tst_Schema.Address.tag>
<tst_Schema.Address.street type='t_Field'>
 <attributes>
  <label>Street Address</label>
  <length>32</length>
  <type>db_String</type>
 </attributes>
</tst_Schema.Address.street>
<tst_Schema.Address.city type='t_Field'>
 <attributes>
  <default>Oshawa</default>
  <label>City</label>
  <length>12</length>
  <type>db_String</type>
 </attributes>
</tst_Schema.Address.city>
<tst_Schema.Address.state type='t_Field'>
 <attributes>
  <default>ON</default>
  <label>Province or State</label>
  <length>3</length>
  <type>db_String</type>
 </attributes>
</tst_Schema.Address.state>
```

```
 <tst_Schema.Address.country type='t_Field'>
  <attributes>
    <default>Canada</default>
    <label>Country</label>
    <length>12</length>
    <type>db_String</type>
  </attributes>
 </tst_Schema.Address.country>
 <tst_Schema.Address.pcode type='t_Field'>
  <attributes>
    <label>Postal Code</label>
    <length>8</length>
    <type>db_String</type>
  </attributes>
 </tst_Schema.Address.pcode>
</tst_Schema.Address>
<tst_Schema.Phone type='t_Class'>
 <attributes>
   <key>
    pid
    tag
   </key>
 </attributes>
 <tst_Schema.Phone.pid type='t_Field'>
  <attributes>
    <label>Id</label>
    <type>db_Integer</type>
  </attributes>
 </tst_Schema.Phone.pid>
 <tst_Schema.Phone.tag type='t_Field'>
  <attributes>
```

```
    <label>Tag</label>
    <length>10</length>
    <type>db_Enum</type>
    <value>
     Home
     Work
     Mobile
     Facsimile
     Other
    </value>
  </attributes>
 </tst_Schema.Phone.tag>
 <tst_Schema.Phone.country type='t_Field'>
  <attributes>
    <default>1</default>
    <label>Country</label>
    <length>2</length>
    <type>db_Integer</type>
  </attributes>
 </tst_Schema.Phone.country>
 <tst_Schema.Phone.area type='t_Field'>
  <attributes>
    <default>905</default>
    <label>Area Code</label>
    <length>4</length>
    <type>db_Integer</type>
  </attributes>
 </tst_Schema.Phone.area>
 <tst_Schema.Phone.number type='t_Field'>
  <attributes>
    <label>Phone Number</label>
```

```
    <length>12</length>
    <type>db_Integer</type>
  </attributes>
 </tst_Schema.Phone.number>
 <tst_Schema.Phone.extension type='t_Field'>
  <attributes>
    <label>Extension</label>
    <length>8</length>
    <type>db_Integer</type>
  </attributes>
 </tst_Schema.Phone.extension>
</tst_Schema.Phone>
<tst_Schema.Electronic type='t_Class'>
 <attributes>
   <key>
    pid
    tag
   </key>
 </attributes>
 <tst_Schema.Electronic.pid type='t_Field'>
  <attributes>
    <label>Id</label>
    <type>db_Integer</type>
  </attributes>
 </tst_Schema.Electronic.pid>
 <tst_Schema.Electronic.tag type='t_Field'>
  <attributes>
    <label>Tag</label>
    <length>10</length>
    <type>db_Enum</type>
    <value>
```

```
     Home Email
     Work Email
     Web Site
     Other
    </value>
   </attributes>
  </tst_Schema.Electronic.tag>
  <tst_Schema.Electronic.data type='t_Field'>
   <attributes>
    <label>
     Data
     Info
    </label>
    <length>32</length>
    <type>db_String</type>
   </attributes>
  </tst_Schema.Electronic.data>
 </tst_Schema.Electronic>
 <tst_Schema.Images type='t_Class'>
  <attributes>
   <key>file</key>
  </attributes>
  <tst_Schema.Images.file type='t_Field'>
   <attributes>
    <label>File Number</label>
    <type>db_Intkey</type>
   </attributes>
  </tst_Schema.Images.file>
  <tst_Schema.Images.tag type='t_Field'>
   <attributes>
    <label>Caption</label>
```

```
      <length>32</length>
      <type>db_String</type>
    </attributes>
   </tst_Schema.Images.tag>
   <tst_Schema.Images.thumb type='t_Field'>
    <attributes>
      <geometry>
       200
       200
      </geometry>
      <label>Image</label>
      <type>db_Image</type>
    </attributes>
   </tst_Schema.Images.thumb>
  </tst_Schema.Images>
```

Appendix B: API Definitions

The Q-Schema API (itm)

TBD.

The Repository API (rpo)

TBD.

The Tuple API (tpl)

TBD.

The Table API (tbl)

TBD.