Data Retrieval and Documentation Using Unity in a UniVerse® Environment

Jose Jimenez

In partial fulfillment of an Honors degree in the department of Computer Science

Ramon Lawrence

December 12, 2002

All requirements for graduation with Honors in the department of Computer Science have been completed.

Douglas Jones

ABSTRACT

Thesis: Access to UniVerse® databases using ODBC clients and within the UniVerse environment can be improved using Unity and its methods.

UniVerse is one of many different types of databases. Poor naming conventions may make the retrieval of information exceedingly difficult. Unity solves these types of problems by using a method of assigning semantic names to each data element in a database. This semantic name acts as documentation of the data element and aids in building queries to retrieve the information stored in the database. The algorithm that Unity uses to assign semantic names to data elements relies on the name presented via an ODBC service. This name, especially within a UniVerse environment, does not encompass all the information available about a data element.

Using Unity in conjunction with programs written within the UniVerse environment, additional documentation can be generated in a semi-automatic way for a UniVerse database. These tools can compensate for poor naming conventions result from UniVerse's separation of the data storage and data access layers. Systems that run under UNIX and other operating systems outside of Windows are required by many businesses; therefore tools were developed to improve access for both ODBC clients and UniVerse programs.

Improvements are made in the amount of information available to Unity using programs written with UniVerse programming tools. These programs take advantage of information stored within UniVerse dictionaries not available to ODBC clients. Some of the methods for interfacing Unity and the UniVerse database are also implemented in UniVerse programs that allow users on the UniVerse host to access information freely. The resulting tools and programs improve access to information in the UniVerse database.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

SECTION 1: INTRODUCTION

Data storage, however well implemented, is worthless without the ability to retrieve that data. While the logical and physical connections may be available, if the data layout is not well documented, it may be near impossible to retrieve the data except through previously developed interface programs. These programs may be limited in their implementation due to static ideas of their use. In order to implement new interface methods, it is necessary to have complete documentation. It may be possible to use an automated approach to document the tables and fields of a database system. This project will explore using Unity [1] and ODBC connectivity to document a large system of tables in a UniVerse [2] environment. It will also explore using a set of host-based programs to generate equivalent documentation. The remainder of this report will examine motivation for the project, the project goals, and the project implementation.

Motivation

In an ideal situation, a company that creates a piece of software would document it and understand it before selling it. In the same ideal situation, a company that creates and sells a suite of software with hundreds of tables would understand and document the complex interactions of these programs and tables. Each column of every table would be documented and, hopefully with little effort, that company would be able to tell which programs required each column. However, not all programmers are software engineers, and not all companies that create software do it by using the appropriate methods. The type of documentation described takes time, and time is money, especially in environments where hourly rates charged to customers are in the hundreds of dollars and projects range in the hundreds of thousands of dollars. Given a limited budget, a company might choose additional functionality over complete documentation and therein lies the root of the problem. When it comes to software systems of any magnitude incomplete documentation is equivalent to no documentation. Any additional changes to the software or database system must be researched heavily, and the only testing that will suffice is full integration with the "live" suite.

Once a company has made a decision for functionality over documentation, it is up to the company programmers and analysts to take up the challenge of documenting the suite of tables and programs and to integrate them with other business tools. Unity is a tool for documenting ODBC-accessible data with X-Specs in a semi-automated manner. Some limitations exist in using an ODBC client to access the data necessary to create X-Specs for the UniVerse environment. Thus it would be preferable to use host tools to create the X-Specs with programs running on the host system, which provide more information about relationships and data than would be accessible to an ODBC client. Once these X-Specs are created, Unity can be used to build data queries to access the data. By documenting tables more precisely using X-Specs, the complexities of future modifications to the system are reduced. Additionally, the information gleaned from creating X-Specs can be used to develop additional data retrieval tools within UniVerse.

Thesis

Access to UniVerse databases using ODBC clients and within the UniVerse environment can be improved using Unity and its methods. Unity provides methods of documenting and accessing multiple data source types. One of the main methods employed by Unity involves improved naming techniques. This methods can be extended to a production system running a UniVerse database. However, there are some problems inherent in the way UniVerse ODBC is implemented that undermine the power of UniVerse dictionaries. Improving documentation with Unity via ODBC or host-based programs will improve access to database information for both programmers and users.

The basic steps required for improving access to the information are:

1) Ensuring the database is accessible to Unity via ODBC.
2) Increasing the number of columns visible to ODBC clients from the UniVerse database.
3) Improving the quality of documentation about the visible columns.
4) Improving data access within UniVerse.

SECTION 2: BACKGROUND

Environment

This project entails examining the use of Unity and X-Specs in conjunction with a UniVerse-hosted database system. Unity is a Windows-based tool for integrating multiple databases. In this situation, the database system will be hosted on an RS6000 system with an AIX operating system (version 4.3). The client system, running Unity, will be a machine running a Windows 2000 professional operating system.

The business environment used in this project is a working database system under UniVerse comprising over 500 data tables with a total size approximating 60 gigabytes. Each of these data tables has anywhere from 0 to over 200 data columns. The number of virtual data columns in each table is not limited in any significant way. The particular business environment being used, like most others, evolves on a day-to-day basis as a result of shifting priorities and business goals. Much of the development takes place on the UniVerse system, so any improvements should be available to users and developers using UniVerse as well as those using Open DataBase Connectivity protocol (ODBC).

Unity is a system developed in part by Dr. Ramon Lawrence currently at the University of Iowa. Unity starts with a global dictionary (GD) that holds information about entities or objects it is likely to encounter within a given database. The GD relates elements of the database to English words. Unity then uses ODBC to access a predefined data source and retrieve data from it. Using standard ODBC interface calls, Unity explores the tables of a database and creates an internal representation of the structure of the database. This internal representation is called an X-Spec. Unity also generates a graphical view of the database. The graphical interface allows the user to enter additional information about the database tables and columns. Unity is then used interactively to assign semantic names to a given database or column in a semi-automated way. Assigning a semantic name improves the documentation for a given column and allows for relating columns from multiple tables and databases using semantic names. Unity can also produce an XML version of an X-Spec. Much like a DTD for an XML document, an X-Spec describes a database and its tables and columns along with all their attributes.

UniVerse is a relational database environment from IBM [3] with built-in ODBC connectivity and its own programming language called UniBASIC. Each database under UniVerse is composed of a number of tables. UniVerse tables are not defined using schemas; therefore they are not constrained to holding a given number of fields at the time they are created. UniVerse separates the data portion of the table from the access portion of the table. In order to link the two aspects of the table, each table has a dedicated table dictionary that defines the fields available for the access layer. In comparison to a relational database system, the dictionary can be seen as a "view" into the table. This table dictionary does not always contain an entry for each field in the table. An entry in the file dictionary contains the information required to retrieve and display the data from the field it pertains to in the table. This information can include formatting instructions, conditional statements, and a label for the data, among other things. Each dictionary item describes a column of the table. Each column available for reporting is a virtual column because of the separation between the access layer and the data layer. These columns can be calculated based on data internal to the table, calculated independently of the data in the table, or based entirely on data from another table. Relational views can only be defined using SQL statements, but dictionary items provide methods for performing joins and other database algebra without using SQL.

There are several different types of dictionary entries. Table 1 provides a sampling of dictionary items for the dictionary of the customer table in the sample used.

Dictionary Record ID	col 1	col 2	col 3	col 4	col 5	col 6	col 7	col 8	col 9	col 10
CM.NAME30	S	3	NAME		S				T	20
CSTM.NAME.ADDR	I	TRIM(PNET.INFO&lt;1,2&gt;:' ':PNET.INFO&lt;1,1&gt;)		PNET CONTACT	25L	S				
CUST.NAME	A	0	NAME		S			A;IF 100 # "" THEN 100 ELSE N(NAME)	R	20
EU.CUS.NAME	S	3	CUSTOMER NAME		S				T	30
FNAME	D	8		NAME XREF	6L	M	M7			
FULLNAME	S	2	BILLTO NAME		S			A;01:"*":02 TCM;C;3;3	L	20

Table 1: A sample listing of dictionary records from the customer file.

The columns labeled "col 1" through "col 10" represent attributes of the UniVerse dictionary items. Attributes in UniVerse correspond to columns of a data table. The type of the dictionary entry is determined by the first attribute of the entry (col 1). This type determines the structure of the rest of the entry. Appendix 1 gives a series of structures for the most common dictionary entry types in UniVerse.

The native programming language of UniVerse is UniBASIC, a compiled version of BASIC with extensions used to work with the databases directly. Programmers do not need to know the details of the hashing algorithm used to access the data portion of the table, but they do need to know in what field number the particular piece of information they are working with is stored. Generally, dictionary items are not used within UniBASIC programs. Along with UniBASIC, UniVerse also utilizes several procedural languages for automating data flow and user interfaces. UniVerse also provides a powerful query language for generating reports. This query language is called RETRIEVE [4]. Programmers can design and execute RETRIEVE reports within UniBASIC by painstakingly concatenating strings that define a RETRIEVE statement using dictionary items. There is no built in utility in UniVerse to assist in this process. Along with RETRIEVE, an SQL variant is also available within UniVerse for executing local queries. This SQL variant is used by the ODBC server to fulfill client requests. These tools make the UniVerse environment a powerful tool for building enterprise software.

In a UniVerse environment, there is a special entry in the dictionary for each table that determines what entries, and therefore what data, are accessible using ODBC. This dictionary item is named "@select." UniVerse SQL relies on the @select entry to describe what columns are accessible, the same way that the ODBC service does. This entry in the file dictionary is generally updated manually. Figure 1 displays the architecture of this environment.

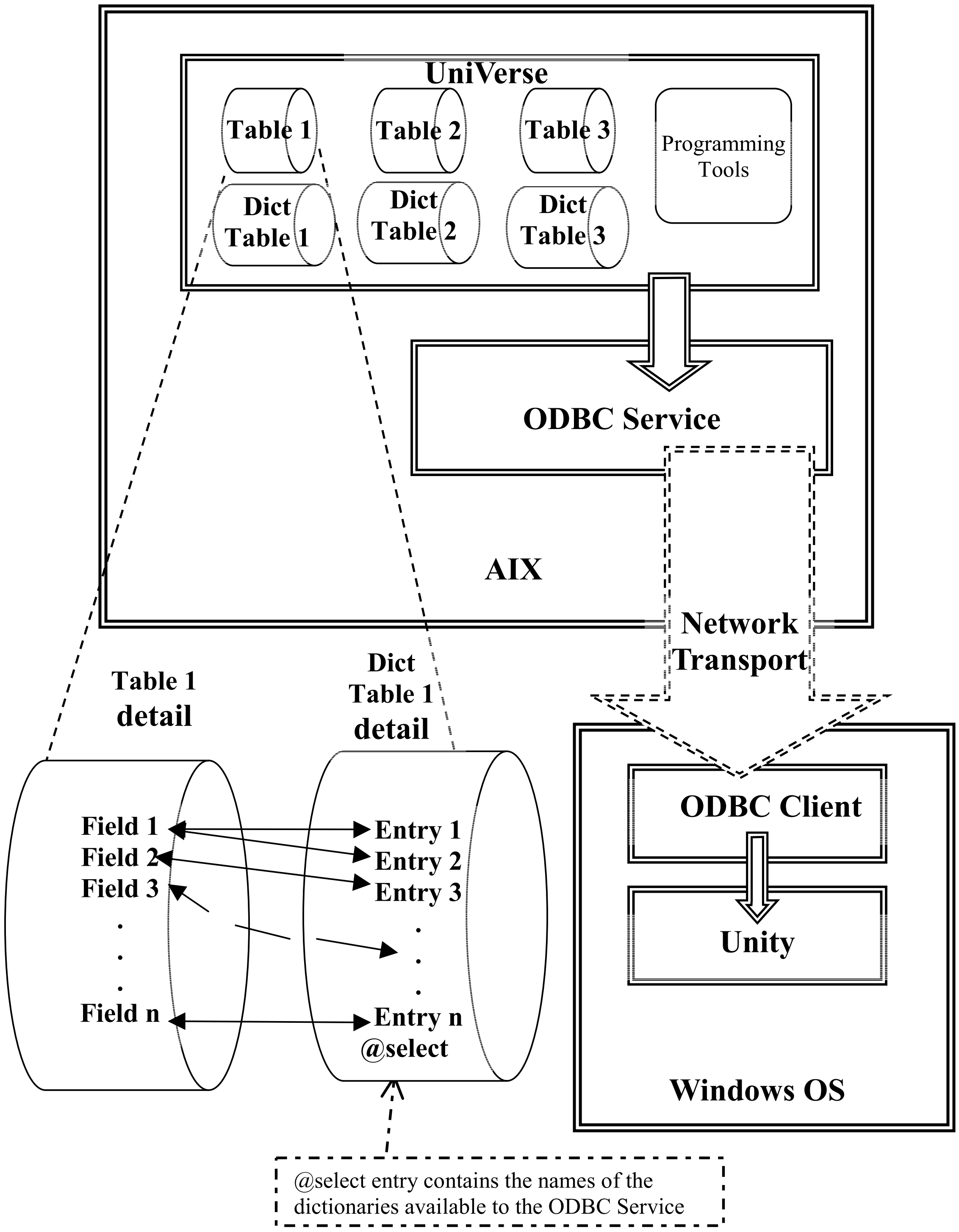

Figure 1: The UniVerse to Unity connection. Note that Field 1 has two dictionary items, Entry 1 and Entry 2, that reference it to produce a value.

Approaches

The goal of this project can be reduced to four subgoals. The first subgoal of the project is to ensure the database is accessible via ODBC. ODBC access is required in order to use Unity's documentation power. The second subgoal of the project is to increase the number of fields accessible to Unity. Increasing the number of fields visible will increase the data fields that Unity can document and access. The third subgoal is to improve the quality of documentation about the visible columns. This improvement will assist in development and data selection. The fourth subgoal is to improve access to the data from within the UniVerse environment. Improving access from within UniVerse is necessary because many systems, such as the business environment being used in this project, still support business goals using the UniVerse environment not only to store the information required, but also as an interface for its users.

Once access is tested using MSQuery, which is closely tied to the Windows OS ODBC client, it will be necessary to increase the number of fields available to Unity. A simple approach to increasing the number of columns available to an ODBC client from a UniVerse database is to include every dictionary item for each table in the @select entry for that table. This "brute force" method, while simple, is not advisable. There are many items that, whether valid for reporting within UniVerse or not, do not work with ODBC and SQL. One example of an item that is valid within UniVerse but not for ODBC is a right-justified column that includes non-numeric data. ODBC tries to define this type of column as numeric because of the right justification. When accessing this type of column, an ODBC client would reject the information because the information passed to it is non-numeric. This problem stems from the implementation of the ODBC service for UniVerse, not ODBC itself. Rather than simply add every dictionary item to the @select entry for the table, a different approach must be found in order to increase the number and quality of columns available to ODBC clients.

The name of the columns visible to an ODBC client is the name of the record within the dictionary for the table. Within UniVerse, this name is generally only visible to programmers using it within a procedural language to generate reports, and the name is not used for programming. The actual label for a column on a report generated with RETRIEVE or one of the other reporting tools available within UniVerse is the column header stored within the dictionary record. Because of this separation between the name of the item and the displayed column header, names for the dictionary items tend to be arcane abbreviations at best and numbers at worst. In order to improve an ODBC client user's experience, more information than is present within the name of a column is required. One approach would be to create additional dictionary items that would be named with the column header.

This approach would improve the readability of the column names but would be limited in its use of the available information for documenting the source of the column. In particular, this project evaluates the performance of two approaches of documenting data layouts in the host database. In the first approach to documentation, the capture process of Unity is used to build X-Specs for the data tables. The second approach requires new programs to be written in UniBASIC. These programs use the information available from within UniVerse to build the X-Specs and retrieve the additional data from the dictionary tables themselves. Improving access from within UniVerse is accomplished by creating UniVerse programs to implement some of the methods developed in the first three subgoals. These tools improve access within the UniVerse environment to both users and developers.

SECTION 3: IMPLEMENTATION

The main goal of this project is to use Unity and some of its methods to improve access to data stored within a particular UniVerse database. While improving access for Unity will improve access for other ODBC clients, some tools should be built to simplify query generation and execution from within the UniVerse environment as well.

Communication between Unity and UniVerse

The first step toward using Unity to build and execute queries and to extract information about the UniVerse database is to ensure this data is accessible to Unity via ODBC. In this stage of the project, a test environment with a limited number of tables is used to test communication between Unity and UniVerse. The tables chosen for the test environment are product and customer information files. In order to test accessibility via ODBC, MSQuery is used in conjunction with Microsoft Excel to access data from the test database. This test gives a baseline for whether or not the UniVerse database is visible to a standard ODBC-based query. To complete the setup of the test environment, some of the fields available to the ODBC client were modified manually to alleviate the field data type problems.

The original version of Unity connects to the data source, but fails to read the table and field information via the ODBC connection. One of the causes for this is the close tie between Unity and Microsoft Access databases. In development and testing of Unity, Access databases are used most often and because of this, some of the source code for Unity is written specifically for Access. This causes Unity to mistakenly report that it is looking for a file with “mdb” as its extension, which is the file extension for Microsoft Access databases. This is a minor issue and is remedied quickly.

A more pervasive issue is Unity's use of ODBC level 3 calls to the data source. ODBC level 3 is the current release level for ODBC drivers. The ODBC driver in use with the UniVerse test environment is only level 2 compliant. This disparity causes some of the data layout discovery features of Unity to fail. Unfortunately, this failure means that one of the basic building blocks of Unity data access, the source definition file, can not be completed. In order to correct the problem in the data source discovery, a new version of Unity uses compatible calls. The new version of Unity can access the UniVerse information and create a source definition file.

With the data source file built, ensuring the rest of Unity works with the UniVerse database is the next priority. Unity is able to build a specification from the source file and define a schema from the specification file. Initially, only one data file is defined with any amount of detail, in order to continue testing. The next connectivity test is to execute a query using Unity against the UniVerse database. A new version of Unity with further modifications is generated at this time to replace further ODBC level 3 calls that execute the generated queries. The modifications are required because the earlier version of Unity does not use the information discovered during the source discovery when executing the query and unnecessarily performs field information retrieval before accessing the data from the tables. In order to get beyond this problem, a further version of Unity replaces ODBC level 3 calls with similar calls conforming to the ODBC level 2 standard.

With Unity able to execute queries against the UniVerse database, it is possible to see some of the power of Unity at work. Unity is able to extract information about the tables and columns visible via ODBC. With the source database presented in Unity's graphical user interface, it is possible to use Unity's semi-automated naming system to assist in documenting the contents of each table and field in the test set. Developing a query with Unity's interface is much simpler now that the names of the tables and columns presented to the user are improved.

UniVerse: Increasing Visibility to the Outside World

The test environment selected initially makes very few columns visible to the ODBC client. More of the columns existing on the UniVerse tables need to be visible to Unity to improve access to the information and allow Unity to assist in documentation. The information required to increase the number of columns visible is not available via the ODBC interface and is only available to UniVerse programs via the table dictionaries. The task of selecting valid entries from those available in the dictionary file is complicated by the fact that dictionaries can be any of four different types. The dictionary records in Table 1 from the customer table in the UniVerse database all relate to the name of the customer but produce different information in various ways.

`ODBC.DICT.CHECK`, written in UniBASIC, reads the dictionary of a table and differentiates between the different dictionary types to parse the dictionary into its component pieces. Using the information found, it can then update the @select entry or generate a report giving details of valid and invalid dictionary entries.

The program can validate the entries in the dictionary file and generate a report or update the @select entry when it executes. When the program works on a given file in update mode, it makes a backup of the previously used data. It also writes a signature to the @select entry of the dictionary file, including the name of the program that updated the file, the date, and the time of update. When adding dictionary items to the @select entry, it copies the entries to new names that have more information than would otherwise be available. The naming convention it uses is composed of several pieces of information separated by a "@", which is converted by some ODBC clients to an underscore. These pieces of information are the base field number if available, the original name of the field and the text label used for reporting purposes. This compilation results in fairly long names, but it is usually easier to tell what the field contains using these new names as compared to using the original names. The names themselves act as documentation of the source of the column.

In order to filter the list of available dictionary entries to valid entries only, `ODBC.DICT.CHECK` has several routines for validation. For example, when parsing dictionaries, right-justified fields merit further investigation since it is impossible for the program to tell simply from looking at the dictionary item what kind of information is actually in the data file. In order to determine if the field suffers from having mixed alpha and numeric data, a sampling of the data is taken to determine if there are any non-numeric values in it.

`ODBC.DICT.CHECK` improved the number and names of the columns available via ODBC. Unity's naming algorithm should be modified to use the additional information provided within the name of the column in applying the built in naming algorithm. This improves its usefulness in documenting column contents with a semantic name. An ODBC client would not have been able to do the work completed by `ODBC.DICT.CHECK` because of its lack of access to information stored within the table dictionaries.

X-Specs: Decreasing the Effects of Increased Information Availability

While increasing the amount of information available to ODBC and Unity, `ODBC.DICT.CHECK` does change data in the UniVerse environment. The dictionaries created are visible to users in the host environment and the names are just as difficult to read without a mapping of the name. `ODBC.DICT.CHECK` increases availability and reliability of fields via ODBC, as described, but it does little to increase the amount of documentation available for a given table that has not been set up for ODBC access. There are situations when it is not desirable or practical to have columns accessible via ODBC. In these situations, to document the tables and columns on the host system, it is necessary to rely on the host system.

Unity uses X-Specs as a method of storing database layout information in a format that is easily shared. The basis for X-Specs is XML, eXtensible Markup Language. This allows the database layout information to be stored in what is, essentially, a text document. However, the meaning of each piece of text and the relationship between pieces of text is determined within the document. An XML document layout can be defined by a DTD, Document Type Definition. Figure 2 shows a partial DTD for an X-Spec.

```
<!ELEMENT XSPEC (System_Name,...,TABLE*)>

<!ELEMENT System_Name (#PCDATA)>
...
<!ELEMENT TABLE
(System_Name,Semantic_Name...,FIELD*,KEY*,JOIN)>
<!ELEMENT System_Name (#PCDATA)>
<!ELEMENT Semantic_Name (#PCDATA)>

...
<!ELEMENT FIELD
(Semantic_Name,System_Name,Field_Type,Field_Size,...,Comm
ent,...,Function_String)>
<!ELEMENT Semantic_Name (#PCDATA)>
<!ELEMENT System_Name (#PCDATA)>
<!ELEMENT Field_Type (#PCDATA)>
<!ELEMENT Field_Size (#PCDATA)>
<!ELEMENT Precision (#PCDATA)>
...
<!ELEMENT Comment (#PCDATA)>
...
<!ELEMENT Function_String (#PCDATA)>
<!ELEMENT KEY
(Key_Name,Key_Type,Key_Scope,Scope_Name,FIELDS)>
<!ELEMENT Key_Name (#PCDATA)>
<!ELEMENT Key_Type (#PCDATA)>
<!ELEMENT Key_Scope (#PCDATA)>
<!ELEMENT Scope_Name (#PCDATA)>
<!ELEMENT FIELDS (FIELD_NAME)>
<!ELEMENT FIELD_NAME (#PCDATA)>
<!ELEMENT JOIN (Join_Name,...)>
<!ELEMENT Join_Name (#PCDATA)>
...
```

Figure 2: A partial DTD for an X-Spec.

Of critical note is the `Comment` element of the `FIELD` element. Although this element is not used for program logic within Unity, it can be used to store information that is available to UniVerse and not accessible via ODBC. An important item to include in this field is the information that `ODBC.DICT.CHECK` was previously providing in its naming convention. This information includes the dictionaries' base column number and column header and is critical in understanding the use and meaning of the contents of a given column.

Once the X-Spec is generated, knowledge about the database in question can be disseminated more readily. Unity does a good job of representing information about databases, but does not have a methodology for importing X-Specs. It was only able to export them.

`ODBC.DICT.CHECK` had previously been used to filter through the dictionary entries for a set of files. This program was modified to create an XML matching the DTD for X-Specs that contained all of the information necessary and available for documenting the UniVerse database being checked. Because XML is at its base a simple ASCII file, this was a simple matter of matching the output of `ODBC.DICT.CHECK` with the DTD.

Now that `ODBC.DICT.CHECK` produces the X-Spec, Unity has been modified to import an X-Spec that conforms to the DTD. In order to import X-Specs into Unity, each of the relevant objects within Unity now has a new method. `CSpec`, `CSpecTable`, `CSpecField`, `CKey`, and `CJoin` all have `XMLExport` and `Serialize` methods; for each object, an `XMLImport` method now combines some of the features of both the `XMLExport` and `Serialize` methods. In order to accomplish this, several string-handling methods are implemented in the new version of Unity. Among them are a function to extract the tag name for a given XML attribute and another to extract the value of the XML attribute. `CKey::XMLImport` in Appendix 2 shows the basis on which the rest of the modifications were modeled. The X-Specs created by `ODBC.DICT.CHECK` allow for documenting the tables and columns not accessible via ODBC in exactly the same way as those that are.

Data Retrieval: UniVerse beyond Unity

Unity can access data available via ODBC, but there are several reasons why UniVerse data may not be available via ODBC. One reason is UniVerse ODBC client licenses are not free. Thus, it is necessary to be able to extract UniVerse data from within UniVerse easily. For a developer working in the UniVerse environment, this extraction is possible by writing programs. However, each separate data extraction program would have to be researched and developed independently of each other.

UniVerse has a built-in tool called `UV.EXPORT` [5]. This tool generates files in either comma-separated ASCII or Lotus Notes format by using the dictionary entries of a table to define columns of the exported file. There are many limitations for `UV.EXPORT`. For example, `UV.EXPORT` does not allow for searching for a dictionary item. Some dictionaries within the test environment hold hundreds of column definitions. With `UV.EXPORT`, searching for a particular dictionary is a matter of searching through an unfiltered list of every record in the dictionary. Poor naming practices often result in problems identifying the appropriate column for an extract. As stated before, a name such as "4" means very little to a user. The column header information is not displayed by `UV.EXPORT`, so this additional information is not available to a user when defining an extract. Additionally, extracting multi-valued data causes the program to generate a file with variable record layouts that is difficult to read and parse. A given datum might fall within the third column on every line unless the second datum was multi-valued. In that situation, it might fall in the third column if the second datum had only one value or the fourth column if the second had two values, and so on.

A better tool would be able to:

1) Create files in comma- or tab-delimited format.
2) Generate XML documents.
3) Select specific rows from a source table.
4) Allow users to build export definitions easily.
5) Allow advanced users to define virtual columns.
6) Handle multi-valued data.
7) Emulate SQL's Group-By feature.
8) Leave room for future development.

In designing the described tool, flexibility is important. Flexibility is achieved by splitting the tool into two logical parts. The first part is the user interface, `BPW.FEXP.UPDT`. This interface allows the definition of the relationship between a source file and an export file. The second part, instantiated by `BPW.FILE.EXTRACT`, is the workhorse of the tool. It interprets the details of the relationship between a source file and an export file created by BPW.FEXP.UPDT and generates the appropriate export file.

Defining an Extract Job within UniVerse

`BPW.FEXP.UPDT` has a built-in storage method that uses standard UniVerse files. Using these methods, it is possible to create and store individual file extraction definitions within user defined groups thereby allowing a user to create logical groupings of file extractions by project, by user, or by any other grouping that is useful. For example, a user can create a group called "monthly" that groups file extractions that have to be executed on a monthly basis. Another example might be a group called "web.products" that would be all the extractions that have to be executed for a product information database hosted online.

`BPW.FEXP.UPDT` has a number of options that apply to the entire extraction routine for a given source to extract file relationship. For example, one option allows the data to be extracted to be passed through a filter that would remove ASCII characters under hexadecimal value 20. These control characters are sometimes embedded into data by accident and their removal before writing them to an extract file is useful. Additionally, a filter can be applied to convert characters such as `>`, `<`, or `&` to their web compliant counterparts such as `&gt;`, `&lt;`, and `&amp;`.. This filter is especially useful in generating XML documents that are to be presented on a web page. The type of file to be created is also defined before defining any of the columns to be extracted. The file is always a plain ASCII file; however, the column delimiter can be defined as either a comma or a tab. If an XML file is desired, then the column delimiter is not used, so the same option toggles between the three types of files to be generated. The XML file root tag can be specified. The program also has an option to ftp the file generated to a given machine within a given folder. While this functionality relies on a UNIX-specific FTP program, it can be generalized to work with any ftp program that takes command line parameters.

Another prompt within `BPW.FEXP.UPDT` is for a select statement to be used in selecting the records to be extracted. This statement can be entered manually, but the user can also use functionality built into `BPW.FEXP.UPDT` to build a select statement interactively.. This feature prompts the user for each of the elements of the select statement and makes it easier for a user with limited or no understanding of the language used by UniVerse to create a select statement. `BPW.FEXP.UPDT`'s interface for interactively generating select statements leaves little room for user error; it is possible however, for a user to enter a syntactically correct string that is not semantically what is desired. For that reason, once the user enters or defines the string, `BPW.FEXP.UPDT` executes the string to present the user with the output. An invalid string is most easily detected by its output. The user can then accept the statement or go back and reenter a new statement. Advanced users, on the other hand, can create a program or use other functionality within UniVerse to create a list and enter any string that can be executed at a UniVerse prompt in place of this selection statement. `BPW.FEXP.UPDT` relies on the user to validate the output of the statement. So long as the string generates a list of records to be used for the export, the string can do any additional work required by the user, such as prompting for input of variables.

`BPW.FEXP.UPDT` prompts the user for the name of the dictionary to use for a given column. The option to select from a list in `BPW.FEXP.UPDT` is an improvement over the selection option in `UV.EXPORT`.. If the user enters an "H" to indicate a request for help in selecting the dictionary item to use, then the program prompts the user for a text string that it will search for in the dictionary. For example, entering DESC would filter the dictionaries and only present the user with items that had DESC in the name or the column header. Using the column header as a search element is similar to using the comment field for documentation within Unity because it improves the amount of information available about a column. This improvement leads to faster selection of fields to include on the report. The user also has the option of creating a virtual column by specifying the details that would normally be stored within the dictionary of the table. This virtual column is never written to the table's dictionary and can only be changed from within `BPW.FEXP.UPDT,` which limits the possibility that a dictionary being changed will affect a stored extract job. `BPW.FEXP.UPDT` also allows the definition of the column header to be used for the extract job. This feature allows the user to place a more meaningful header on each row, but it is also used in generating XML file extractions. For XML, the column header entered becomes the element name for the value to be extracted.

Additionally, for each column to be extracted, an advanced user, defined within a record stored in a control file under UniVerse, has the ability to mark each field as a summary field or a multi-valued field. A summary field will be totaled and the detail of each record will be omitted in the extracted file. Each column appearing before a summary field will be treated as a grouping field. For example, given the four fields of Warehouse, Product line, Customer type, and Sales Dollars (with Sales Dollars being a summary field) the extraction program would produce a file with Sales dollars totaled at each change in Customer type. The select statement for this type of extract must match the fields that appear before the summary field or fields. The statement must select the records to be extracted in sorted order by, for the given example, Warehouse, Product Line, and finally Customer type.

UniVerse multi-valued data can be contained within the same record as other data is not associated with it. For example, in a UniVerse order file, a tax code and tax amount may be in the same record as a fee code and a fee amount. If each of these were multi-valued, then there could be two tax codes and corresponding tax amounts and three fee codes with corresponding fee amounts. UniVerse ODBC relies on dictionaries to mark the related fields. When these fields are marked correctly, the UniVerse ODBC server will present these to a client as part of separate tables. For the given example, since no other multi-valued fields are in the file, one table presented to an ODBC client would have all the fields that are not multi-valued within the order file. Two additional tables would be presented each with record identifiers matching the first table. One of these additional tables would have the tax code and tax amount information split into multiple records; the other would have the fee code and fee amounts. `BPW.FEXP.UPDT` uses a similar model in order to accommodate multi-valued fields. In a given extract job, any number of single-valued fields can accompany a single set of multi-valued fields. One of these is marked by the user as the root; in other words, this is the master column used to determine how many fields must be extracted. Continuing with the same example as above, the product number purchased, the product description, the customer number, and other fields may be extracted along with the fee codes and fee amounts. The resulting data will duplicate all the other columns within the record once for each fee code if the fee code is marked as the multi-valued root. The fee code and fee amount columns will have discrete values for each row extracted.

Given the following record, with ellipses marking fields not shown,

Record id	**Customer**	**Product**	**Description**	….	**Fee code**	**Fee amount**
1234567*1	01*000261	17420	Asian Spice Ginger Brew	…..	F1	0.50
					F2	1.25
					F3	0.02

Extracting each of the columns visible yields:

Record id	**Customer**	**Product**	**Description**	**Fee code**	**Fee amount**
1234567*1	01*000261	17420	Asian Spice Ginger Brew	F1	0.50
1234567*1	01*000261	17420	Asian Spice Ginger Brew	F2	1.25
1234567*1	01*000261	17420	Asian Spice Ginger Brew	F3	0.02

It would also be possible to directly emulate the ODBC clients' output by creating a separate extract for each multi-valued field, which would result in an extraction such as:

Record id	**Fee code**	**Fee amount**
1234567*1	F1	0.50
1234567*1	F2	1.25
1234567*1	F3	0.02

along with another extraction that would have all the single-valued data desired such as:

Record id	Customer	Product	Description
1234567*1	01*000261	17420	Asian Spice Ginger Brew

Executing an Extract Job within UniVerse

In order to accomplish the extractions, `BPW.FEXP.UPDT` calls `BPW.FILE.EXTRACT`, which is a separate UniVerse program, with command line parameters describing the work be completed. Some of the parameters to `BPW.FILE.EXTRACT` are

OPTION	Description
REPORT	Indicates a single extraction is to be completed
PREFIX:groupid*[recordid]	Indicates which group or individual job is to be done
LIST:listname	Indicates a list of record id's to use in place of the select statement entered using BPW.FEXP.UPDT

`BPW.FILE.EXTRACT` was written as a separate tool to allow a variety of uses. A call to `BPW.FILE.EXTRACT` can be embedded in any UniVerse program, allowing a programmer to design a standard report using `BPW.FEXP.UPDT`. A program can then be written to retrieve selection options from a user for a specific task — for example, getting the customer number and start and end dates for a report of ordering activity and generating a list of matching record identifiers. This list can then be passed to `BPW.FILE.EXTRACT` along with the group and record identifier of the report desired. This process saves programming time when the desired output is a file rather than a printed report. Although the concept of a "paperless office" has lost ground due to its seeming impossibility, being able to manipulate data using programs such as COGNOS and other data analysis tools is a definite and achievable improvement.

`BPW.FILE.EXTRACT` has additional options that match up with options in

BPW.FEXP.UPDT. A protocol for date substitutions has been incorporated that allows a user or developer to create a report that does not need to be continuously modified in order to continue to be useful. Again, using the orders file as an example, a user might want a report that shows what orders are going to ship today. Rather than enter a date in the selection prompt such as “12/10/02”, the user would enter the statement that would select the data as required but enter “(RW_FDT)” where the date would normally go within BPW.FEXP.UPDT. BPW.FEXP.UPDT would accept that as a valid date and BPW.FILE.EXTRACT would replace the text within the select statement with the given date, “12/10/02”. Some of the date replacements available are:

Text	**Meaning**
(RW_FDT)	Replace With Formatted DaTe
(RW_YFDT)	Replace With Yesterday’s Formatted DaTe
(RW_RDT)	Replace With Raw DaTe
(RW_YRDT)	Replace With Yesterday’s Raw DaTe

UniVerse uses a base date system, which internally represents any given date as the number of days from the base date. Using this type of system, a developer, if not a savvy user, can select data for any given range of dates by using evaluated virtual fields in the select statement. The use of this type of select statement is beyond the scope of this paper, but the flexibility is apparent.

In order to determine the value for a given column, BPW.FILE.EXTRACT has to build on some of the strengths of UniVerse. It uses a straightforward algorithm that accomplishes much of what UniVerse’s RETRIEVE tool does. Figure 3 shows a partial decision tree for the algorithm. See Table 1 for examples of dictionary records.

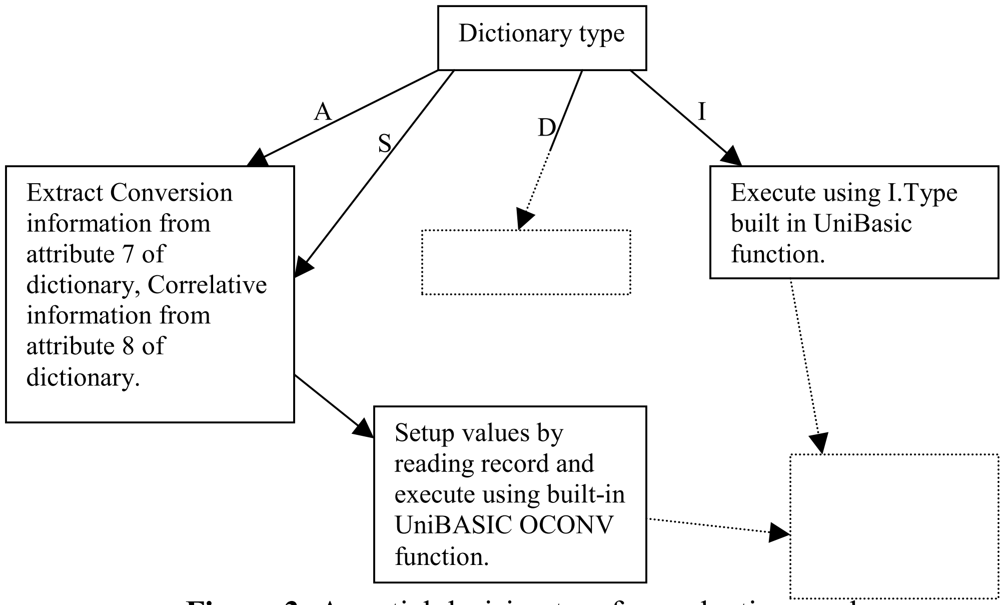
Dictionary type
A
S
D
I
Extract Conversion information from attribute 7 of dictionary, Correlative information from attribute 8 of dictionary.
Execute using I.Type built in UniBasic function.
Setup values by reading record and execute using built-in UniBASIC OCONV function.

Figure 3: A partial decision tree for evaluating a column.

The decision tree does not show some of the steps required to evaluate columns that reference other columns. In order to evaluate these columns, it is necessary first to de-reference any named columns into their respective values and then continue to evaluate the desired column using these values.

Because of this reporting tools flexibility, it is possible to set up a CGI[1]-based interface for `BPW.FILE.EXTRACT` that lists all of the defined groups and the defined reports within each group. This list is presented to users via an intra-net web site; clicking on any of the listed groups will execute that group and present hyperlinks to the data. One of the hyperlinks presents the data as an HTML table. The other is a hyperlink that can be used to download the data in its chosen format, whether XML, tab-, or comma-delimited.

[1] **C**ommon **G**ateway **I**nterface – one method of adding a program or script to a web page to interact with resources not normally accessible to it, but available to the server.

The UniVerse programs created for this project directly access UniVerse data that may not be available via ODBC. Similar to SQL views, an extract job or group defines a view into the database that allows the definition of virtual fields, which are not in danger of being changed outside of the extract interface. The programs take advantage of information not accessible to ODBC clients, such as the column header of the dictionary. Developers or users can easily change the content of an extract job without having to sift through code or modify any programs. The ease with which extract jobs can be changed decreases the amount of time it takes to get information out of the database.

SECTION 4: RESULTS

Before executing `ODBC.DICT.CHECK` to update dictionary items, the file used to test ODBC connectivity had 57 columns available for queries via ODBC and SQL. Once `ODBC.DICT.CHECK` was executed selecting the update option, 521 columns were available for queries. The additional columns represented dictionary items accepted by `ODBC.DICT.CHECK` as valid for ODBC and SQL access. A simple increase in the number of columns available is not necessarily an improvement if there is no improvement in the quality of the information available. The benefit of the additional columns and the improved naming methodology became apparent when accessing the fields using Unity. Deciding on semantic names and whether or not to add these fields to the specification of the data source was much easier with the new data.. Quantifying this improvement was not attempted; however, comparing a label such as "4" on a dictionary item to a label of "O@4@F4@TAXCODE" on the column, suggests dramatic improvement. The first label may hint at the column number from which the data comes. The second label indicates the data in the column corresponds to a TAXCODE. Hence the second label is much easier to work with and assign a semantic name to. Unity's semi-automated naming algorithm can be modified and used to assign a semantic name for a column with this type of name which would have been impossible using the original name of the column.

The new dictionary entries helped to increase access to the ODBC client but still suffered from having arcane names, even given the obvious improvement. A better solution building on the strengths of Unity and UniVerse came from the creation of X-Specs using `ODBC.DICT.CHECK`.. In X-Spec creation mode, `ODBC.DICT.CHECK` did not need to create new dictionaries or even increase access to the UniVerse tables. However, the resulting X-Spec gave a map of the database that had not previously been accessible. The names for the columns continued to be arcane, but the fact that the originals were preserved is important. These names are referenced in the stored reports generated in the RETRIEVE or UniVerse SQL language. The information previously provided within the name of the column created by `ODBC.DICT.CHECK` is now present in a comment field visible within Unity. This added documentation and graphical representation of the information contained within will prove invaluable in documenting the database. It will also allow users to share documentation of their databases without needing to share access to their data.

`BPW.FEXP.UPDT` and `BPW.FILE.EXTRACT` have drastically improved the development time for new data extractions. In general, the programs allow a developer or a savvy user to create a data extraction routine that can be stored and repeated as needed. With the new tool, simple extractions can be designed and completed by users. The extraction can then remain static or be modified quickly and easily by a user or developer without the need to modify or write a single line of code. Previously each data extraction was requested separately from a report request and was completely static when coded by a developer.. Paper reports were the norm for new development. This meant that useful information was trapped on paper where it could not be manipulated easily. With the new tool, using simple FTP protocols, the generated file can be copied to a client system and then manipulated with standard tools such as Excel, Lotus Notes, or any other product that can import ASCII or XML files.

SECTION 5: FUTURE DEVELOPMENT

Unity

Unity should be expanded to make it compatible with more data sources. As discovered when trying to work with the UniVerse database, other types of databases may not be compatible with ODBC level 3 or level 2. Unity could be modified by creating inheritance classes that could be called for different levels of access. This modification would entail creating a master class that would have the calls for ODBC level 1 and subclasses that would override certain functions, such as the data field discovery. Another use for these would be the addition of ODBC-driver specific calls where necessary. For example, UniVerse ODBC clients have two different methods of accessing tables. These two methods cause multi-valued fields to be returned differently. If one method is preferable to another, then opening the data source should be done in such a way as to take advantage of the correct method. This would require different calls that are not necessary for other database types and would best be separated from the rest of the code by creating inheritance classes.

Within Unity, generating an SQL query is done using a graphical interface. This process represents the query in an internal format that is then translated into SQL. Unity could be expanded to generate query strings using various other syntaxes, including RETRIEVE statements that are used within the UniVerse environment. Since the query is represented graphically to a user, the translation of the query into a particular syntax could be a simple matter of mapping one well-structured language to another. The graphical query system may not be compatible with some syntaxes but could be extended by initially signifying what query language was to be generated when beginning the process.

`BPW.FILE.EXTRACT` and `BPW.FEXP.UPDT`

It is possible to create a socket-driven interface for `BPW.FILE.EXTRACT`. To accommodate such future development, `BPW.FILE.EXTRACT` can take a control file as a parameter in place of its PREFIX parameter. Given a file path for a simple ASCII text file with the appropriate information within it, `BPW.FILE.EXTRACT` will open and parse the file for the information it requires to execute an extraction job.

Additional work for this system involves performing joins within `BPW.FEXP.UPDT` in order to make additional information available for any given extract. If access to any field within a given file is already performed within a dictionary item for a file, then access to any field can be granted. `BPW.FEXP.UPDT` and `BPW.FILE.EXTRACT` are already examining dictionary items. To verify that a given dictionary accesses a file other than the one being reported from, `BPW.FEXP.UPDT` must check for the command strings that implement that access. Adding access to any other field is then a simple matter of changing the field number in the dictionary to the requested field and executing the dictionary.

Little attention has been paid to `BPW.FILE.EXTRACT`'s ability to generate XML files. However, it too can offer some improvement. By incorporating a DTD into the XML generated, `BPW.FILE.EXTRACT` would improve verifiability of the data contained within it and would assist a user in sharing the data with another user by describing the contents of the file to a greater extent. This ability to generate XML files could also be useful in providing data via a simple HTML interface. It is possible to write a web service that would execute a file extraction routine and then return an HTML page with the XML embedded within it. Within the .NET framework, for example, a call to this web service could be read with an XML reader. Creating such a service would provide a method for displaying selected pieces of information on a web page from the UniVerse server without having to write additional code on the UniVerse side to accomplish it.

Documentation

The X-Specs generated by `ODBC.DICT.CHECK` provide some information about the database not previously accessible via ODBC or SQL. However, there is more information that can be displayed and presented. For example, displaying whether or not a field is multi-valued is useful. Some of this information can be culled from the dictionaries and other standard documentation. This information could all be added to the comment field of the X-Spec, or the X-Spec could be extended to add attributes to hold some of the more universal data. In order to make Unity a more useful tool for documentation, a mouse-hover event should be added. When the mouse is placed over a column representation in the graphical view, the comment field information should display until the mouse moves. Rather than have a user open the properties window of the column, this method of displaying the information could prove to be critical, especially when names are of little use in deciphering the contents of a field.

One of the major benefits to using Unity to document the information available in an ODBC or otherwise accessible data source is Unity's use of semantic names. The semantic name, once defined, is a user's first point of contact with a given column. If enough information is encoded in the name, the user does not need to look at the comments of a column to determine what it contains. To help improve the automated assignment of semantic names, Unity may be modified to access the information required wherever it is stored in the data source.

SECTION 6: CONCLUSION

The main goal of the project was to increase access and documentation of information stored within a UniVerse database. Using Unity and the UniVerse tools developed, it was possible to increase the accessibility between UniVerse and ODBC clients as well as increase access to the data from within UniVerse. These same tools increased the amount of documentation available for the database. The majority of the tools used, with the exception of the web interface developed for the UniVerse data extraction tool, are applicable to other UniVerse databases.

In pursuing the goal of the project, it became clear that the names of the columns were a critical part of user documentation. The name of the column is what is presented, not only to ODBC clients, but also to a UniVerse user when the contents of a dictionary are listed. While a UniVerse user has the ability to see the column header used with a dictionary item, an ODBC client does not. X-Specs provide a method for accessing information that is not within the name of a column. An X-Spec also provides a comment attribute in which information relating to the column can be kept. Using an X-Spec, a user or a developer can better track information about a column, including a semantic name that gives not only a better name for the field, but also an indication of the relationship between it and other columns throughout a database or a group of databases. The UniVerse tools developed compensated for the poor naming conventions by presenting the column header along with the column name. This feature assisted in development and compensated for the lack of documentation as well. This project improved access to information in UniVerse databases by providing additional tools and documentation using Unity and techniques learned in implementing Unity in a UniVerse environment.

APPENDIX 1: UNIVERSE DICTIONARY ENTRY TYPES AND THEIR STRUCTURES [6]

D-Types

Column	Column Name	Contents	Description
0	@ID	*Name of the dictionary entry*	Identifies the field
1	Type	D	Type code
2	Loc	*Field number*	Location of the field in the data table
3	Conv	*Conversion code*	Formula to convert the data into external format
4	Name	*Column heading*	Name used as column heading
5	Format	*Width and justification*	
6	SM	S | M	Single-valued or Multi-valued flag
7	Assoc	*Phrase name*	A phrase name that links multi-valued fields
8	Data Type	*Data type*	SQL data type if present (Usually not)

I-Types

Column	Column Name	Contents	Description
0	@ID	*Name of the dictionary entry*	Identifies the field
1	Type	I	Type code
2	Exp	*I-Type expression*	Expression that produces values for the field
3	Conv	*Conversion code*	Formula to convert the data into external format
4	Name	*Column heading*	Name used as column heading
5	Format	*Width and justification*	
6	SM	S | M	Single-valued or Multi-valued flag
7	Assoc	*Phrase name*	A phrase name that links multi-valued fields
8	Data Type	*Data type*	SQL data type if present (Usually not)

A-Types & S-Types

Column	Column Name	Contents	Description
0	@ID	*Name of the dictionary entry*	Identifies the field
1	Type	S | D	Type code
2	Loc	*Field number*	Location of the field in the data file
3	Name	*Column heading*	Name used as column heading
4		C;number[;number…]| D;number	Links multi-valued fields
5		NONE	Not used
6	Data Type	*Data type*	SQL data type if present (Usually not)
7	Conv	*Conversion code*	Formula for converting stored data to external format
8	Corr	*Correlative code*	Formula that produces values for a field
9	Typ	L | R | T |U	Justification code for a field
10	Max	*Width*	Column width for a RETRIEVE report

APPENDIX 2: IMPORTING THE KEY DEFINITION PORTION OF AN X-SPEC

```
void CKey::importXML(ifstream &os)
// Imports key information of X-Spec from XML format
//    from open ifstream os
{
     //Key level information
     // basic algorithm:
     //    assumption:  have already read <KEY> tag
     //    read line that contains name
     //    parse value from line
     //    read line that contains type
     //    parse value from line
     //    read line that contains scope
     //    parse value from line
     //    read line that contains scope name
     //    parse value from line
     //    while line is not <FIELDS>
     //       read next line
     //    read next line
     //    while line is not </FIELDS>
     //       parse field name from line
     //       read next line
     //    read next line

     CString line_data = "";
     CString line_tag;
     CSpecField     *sfld;

     line_data = getnextline(os);
     name = getvalue(line_data);

     line_data = getnextline(os);
     type = atoi(getvalue(line_data));

     line_data = getnextline(os);
     scope = atoi(getvalue(line_data));

     line_data = getnextline(os);
     scope_name = getvalue(line_data);

     line_data = getnextline(os);
     line_tag = gettag(line_data);
     while (! (line_tag == "FIELDS"))
     {
          line_data = getnextline(os);
          line_tag = gettag(line_data);
     }

     line_data = getnextline(os);
     line_tag = gettag(line_data);
     while (! (line_tag == "/FIELDS"))
     {
          sfld = new CSpecField;
          sfld->sys_name = getvalue(line_data);
          AddField(sfld);

          line_data = getnextline(os);
          line_tag = gettag(line_data);
     }

     line_data = getnextline(os);

}
```

REFERENCES

[1] R. Lawrence and K. Barker: Unity - A Database Integration Tool. TRLabs Emerging Technology Bulletin December 4, 2000.
[2] IBM: IBM Software: Database and Data Management: U2 product family: UniVerse: Overview. http://www-3.ibm.com/software/data/u2/universe/
[3] IBM: IBM Software: UniVerse System Description. http://www-3.ibm.com/software/data/u2/pubs/library/952univ/70-9003-952.pdf
[4] IBM: IBM Software: UniVerse Guide to RETRIEVE. http://www-3.ibm.com/software/data/u2/pubs/library/952univ/70-9005-952.pdf
[5] VMARK: VMARK Technical Bulletin: Part No. 74-0074: UVEXPORT: The UniVerse Export Facility. VMARK Software Inc., 1993.
[6] VMARK: UniVerse System Description: Part No. 70-9003-931. VMARK Software Inc., 1996.