11-2

SANDIA REPORT

SAND95–2409 • UC–900
Unlimited Release
Printed November 1995

The Web Interface Template System (WITS), A Software Developer's Tool

Lois J. Lauer, Mark Lynam, Tammie Muniz

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

SF2900Q(8-81)

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A05
Microfiche copy: A01

SAND95-2409
Unlimited Release
Printed November 1995

Distribution
Category UC—900

The Web Interface Template System (WITS), A Software Developer's Tool

Lois J. Lauer
Mark Lynam
Tammie Muniz

Financial Systems Department
Thomas L. Ferguson, Manager
Sandia National Laboratories
Albuquerque NM 87185

Abstract

The Web Interface Template System (WITS) is a tool for software developers. WITS is a three-tiered, object-oriented system operating in a Client/Server environment. This tool can be used to create software applications that have a Web browser as the user interface and access a Sybase database. Development, modification, and implementation are greatly simplified because the developer can change and test definitions immediately, without writing or compiling any code. This document explains WITS functionality, the system structure and components of WITS, and how to obtain, install, and use the software system.

Acknowledgment

The WITS development team thanks Andrea Cassidy for her help in authoring this document. She kept focus on the reader's needs and point of view, and on what we intended to accomplish with the document structure. She worked, through many iterations, to keep the document user friendly. This document is essential to the reusability of the software, and the team appreciates Andrea's willingness to volunteer for this task.

We also thank Mary Roehrig, who not only volunteered to do usability testing on the GUI tool, but also made changes in the WITS Template Builder to improve the application.

Thanks to our manager, Tom Ferguson, for his support of integrated development of 3-tiered applications and reusable code, as well as providing the resources needed to implement this software in the short period of time allowed as part of the CIO's 1995 May Deliverables.

Contents

Figures

1. Summary

This document describes the Web Interface Template System (WITS), a software developer's tool. WITS is a product that can simplify the development of software applications and speed software delivery. This tool can be used to create applications that retrieve data from the Sybase data warehouse and present it to the end user via a Web browser or other interface. Data can also be inserted, updated or deleted via Sybase stored procedures. WITS dynamically creates and executes Structured Query Language (SQL), or accesses stored procedures, as defined in data templates. Web pages are displayed back to the browser by combining the retrieved data with HTML templates, according to embedded substitution tags. These template definitions can be reused both within and across applications. HTML templates can include symbolic hypertext links which contain the data values retrieved and point to other templates. This allows the application end user to drill down to lower levels of detail or link to data related to another WITS application.

Applications developed using WITS provide these features:

- applications are available to all Internal Web clients without installation
- updates to applications do NOT require any client maintenance
- generic functions replace the need to write and compile your own programs
- application definitions are input through a graphical user interface (GUI)
- testing application definitions requires only a "reload" in the Web browser
- data definitions can insulate the functions from physical database changes
- a parameter driven environment allows for ease of implementation
- activity logging provides the ability to monitor system usage

This document is written for software developers who will be creating software applications that have a Web browser as the end user interface and access a Sybase database. It can also be used to develop a non-Web interface to the data retrieval portion of the system.

- The first part of this document explains the concepts and functionality of WITS.
- The second part is a "cookbook" or list of steps the developer would follow to design and create a software application using WITS.

WITS functions were written in the C language using Sybase Open Client DB-Library and run on a UNIX platform. The GUI interface (WITS Template Builder) was developed in PowerBuilder™ 4.0 and runs on a PC Microsoft Windows™ platform. It uses Sybase Open Client Client-Library and the PowerBuilder 4 Deployment Kit. Developers must have a knowledge of HTML, basic UNIX, and some experience with data retrieval using SQL to use WITS in their Web software projects.

WITS was developed as part of the "Financial Queries on the Web" project, with the intent to produce reusable code whose functionality would be applicable to more than one project.

2. WITS Advantages and Functionality

There are many different ways to display data on the Web, however the generic WITS functions were developed to provide the following advantages:

1. an alternative to writing your own 3GL code for:
   - data retrieval from Sybase databases
   - dynamic intermingling of Hyper Text Markup Language (HTML) and retrieved data (WITS does this with Substitution Tags)

2. simplified modification and testing of your application:
   - make and test changes in a GUI environment, without recompiling code
   - use verbose mode to trace and debug as your application executes
   - run-time DBMS error logging

3. parameter driven environment for ease of migration:
   - Universal Resource Locator (URL) paths
   - dynamic broadcast of banner messages to application users's screens
   - Sybase environment values
   - Sybase login time-out and SQL time-out

4. the ability to leverage the power of Sybase stored procedures:
   - to create, update, delete and retrieve data from Sybase databases
   - to create complex queries not supported by single table or view access

5. logging of application activity by template ids

2.1 An alternative to writing your own 3GL code

Developers will not need to write any SQL (or any C or other 3GL code) for their Web application. WITS dynamically generates and executes all SQL statements, based on the template definitions that the developer created using the WITS Template Builder. Consequently, professional-quality applications can be developed by individuals who are not familiar with the syntax of SQL.

For applications that do not use a Web browser interface, the developer will need to write code for an interface to pass data into and receive data from the WITS Query Retrieval Processor (QRP).

If the developer wishes to use the stored procedure capability of WITS, then a working knowledge of Sybase stored procedures is needed. It is strongly suggested that developers wishing to create applications using WITS be familiar with SQL and with the database structure they are querying.

A knowledge of HTML is required for Web applications, since the developer must write all necessary HTML. HTML is not regarded in this document as a Third Generation Language (3GL).

Additionally, a summary knowledge of UNIX is required to create and maintain the shell scripts and access files for each application. Primarily this means the developer should have either some experience using vi or some other editor for UNIX files.

2.1.1 No code needed for Data Retrieval from Relational Databases

New databases and tables can be referenced and accessed simply by defining the specifics for connectivity, database, table, and columns to the WITS GUI Front End, the WITS Template Builder. The WITS Template Builder stores and maintains (in Sybase template tables) all information needed to access the target database. This means that if developers need to change the names of databases, tables, or table columns of the target database they can do so through the WITS Template Builder, without application code changes.

The target database supported by WITS in its initial deployment was developed to support only the Sybase System 10 DBMS. In the future it will be able to support retrieval from other relational databases (see section 12, WITS Future Enhancements).

2.1.2 No Code needed for Dynamic Intermingling of HTML and Retrieved Data

The actual placement of returned data into the HTML page is done with a WITS-specific series of characters known as the substitution tag. In addition to displaying retrieved data to the user, substitution tags can be used to pass data from screen to screen, even though that data has no involvement with an underlying query. For the format and details of these substitution tags, see section 4.1.2, Substitution Tags.

2.2 Simplified modification, testing, and debugging

The developer can quickly make and test changes to a Web application by keeping both the browser and WITS Template Builder windows open simultaneously. The actual definition of an application's behavior is found in the Sybase template tables, which the developer creates and maintains through the WITS Template Builder. This allows the developer to change any portion of an application's definition in the WITS Template Builder, then immediately test it by simply executing the application via the browser. Although the run-time portion of the functional layer of WITS is resident on a UNIX machine, the presentation layer for development (the WITS Template Builder) and for display (the Web browser) is based in Windows

WITS has a debugging feature known as verbose mode. Verbose mode provides a real time dump of internal information being created when the main functions execute the query and display the query results. This provides the developer with a window into what the application is doing with the templates that were defined using the Template Builder. The developer can then make appropriate adjustments to resolve any problems.

In the verbose mode screen there is a scrollable text box that is loaded with either the SQL "SELECT" statement or stored procedure "EXEC" statement. The developer can easily copy this text to the Windows clipboard and pass it into a database analysis tool such as Rapid SQL for execution. This capability allows the developer to collect statistics about the execution of an SQL statement (that is, to determine if the statement is using an index, and if so, what index) for performance tuning.

2.3 Parameter-Driven Environment for Ease of Migration

WITS was designed so that any value needed to attach to Sybase, as well as other variables to the environment, is not embedded in the application system. Instead these values and variables are parameters passed to WITS from an external source. There are three sources for these parameters: a UNIX Shell Script, a UNIX Access file, and the WITS data_group table.

2.3.1 Parameter source - the UNIX Shell Script

The UNIX shell script is the top-level function in the hierarchy of a WITS application. The UNIX shell script may not be needed for non-Web applications, although it could be used to invoke the interface to QRP. Typically there is one shell script per data group. (An application may access more than one data group.) For a Web application, the shell script contains the path and program name of the WITS Web Template Processor (WTP). The shell script also contains the default template ID, the path and filename for the access file, and the $QUERY_STRING. See section 5.1, UNIX Shell Script with $QueryString, for a full definition of the UNIX shell script command syntax and meaning.

2.3.2 Parameter source - the Access File

The access file is a UNIX flat file. The -a option on the command line of the UNIX shell script is used to pass the full path and filename of this access file to the WITS system. Typically there is one access file for each data group. (An application may access more than one data group). The access file contains a series of mandatory and optional parameters which both the WTP and QRP processes use to perform their tasks. These parameters include all those required to establish connectivity with the query parameters of the database, including login and query execution time-outs. During the initialization phase of WITS run-time execution these parameters are read in and made available to the WTP and QRP. See Section 5.4, Access File with Parameters, for a full explanation of all access file parameters.

2.3.3 Parameter source - the WITS Data_Group Table

All the access file parameters required to establish a connection to the SQL server that contains the data to be queried are also defined in the WITS Template Builder. These parameters are stored in the data_group table.

Although these parameters can be defined in the WITS Template Builder, at this time only the DBNAME parameter is being used. The DBNAME is used for identifying the location of the stored procedures on the current SQL server.

The long-term purpose of the data_group table is to support multiple concurrent DBMS process sessions. See section 12, WITS Future Enhancements and Extensions.

2.4 The ability to leverage the power of Sybase stored procedures

In addition to the dynamic generation of SQL statements to retrieve data from the DBMS, WITS also supports Sybase stored procedures as an alternative and potentially more powerful means of retrieving data. Using stored procedures allows for far more advanced methods of data retrieval than are available through the single table/view approach that is provided by dynamic SQL statement generation. Stored procedures can also be used to enforce input criteria validation by using the RAISE ERROR feature of Sybase. WITS also can be used to create, update, and delete rows from a Sybase database via stored procedures.

2.5 Logging of Application Activity

The purpose of log files is to collect statistics on the frequency and volume of reports being viewed by the end user. This will provide feedback on whether customers are using an application and which templates are most frequently accessed.

3. System Overview - How the parts of WITS work together

WITS is a system of components which can be categorized by purpose. Each component will be used to either define the application to the WITS system, define the environment, or execute the application. The WITS components are used as follows:

1. Defining an application to the WITS system requires:
   - the HTML page definitions (see section 4.1.1 HTML)
   - the GUI Front End to the WITS tables (see section 4.2, GUI Front End)
2. Defining the environment and executing the application requires:
   - the Shell Script (see section 5.1, UNIX Shell Script with $QUERY_STRING and Command Line Parameters)
   - the Web Template Processor (see section 5.2, Web Template Processor)
   - the Query Retrieval Processor (see section 5.3, Query Retrieval Processor)
   - the Access File (see section 5.4, Access File with Parameters)

WITS was designed for developing Web Applications, but the functionality for the Web interface is separate from the data access functionality. This was done to enable other systems to utilize the data access function in non-Web applications. The following figures show the functional parts of a Web application and how non-Web applications can reuse the data access function.

Figure 3.0 —1 shows that for applications with a Web interface, query criteria are passed from a user's Web browser to the WITS Web Template Processor (WTP) common interface. The WTP uses predefined HTML templates created by the developer to convert query criteria and create parameter requests for the WITS Query Retrieval Processor (QRP). The QRP processes the request, sends a command to a Sybase server, and returns a condition code and a data array of query results. The data array and condition code are sent back to the Web Template Processor, which will insert the array values into formatted HTML for the Web browser to display to the user.

Figure 3.0 —2 shows that if a Web interface is not being used, the query request comes into the QRP from another interface such as one built for PowerBuilder, Visual Basic, or some other tool or function, and the results are passed back to that interface.

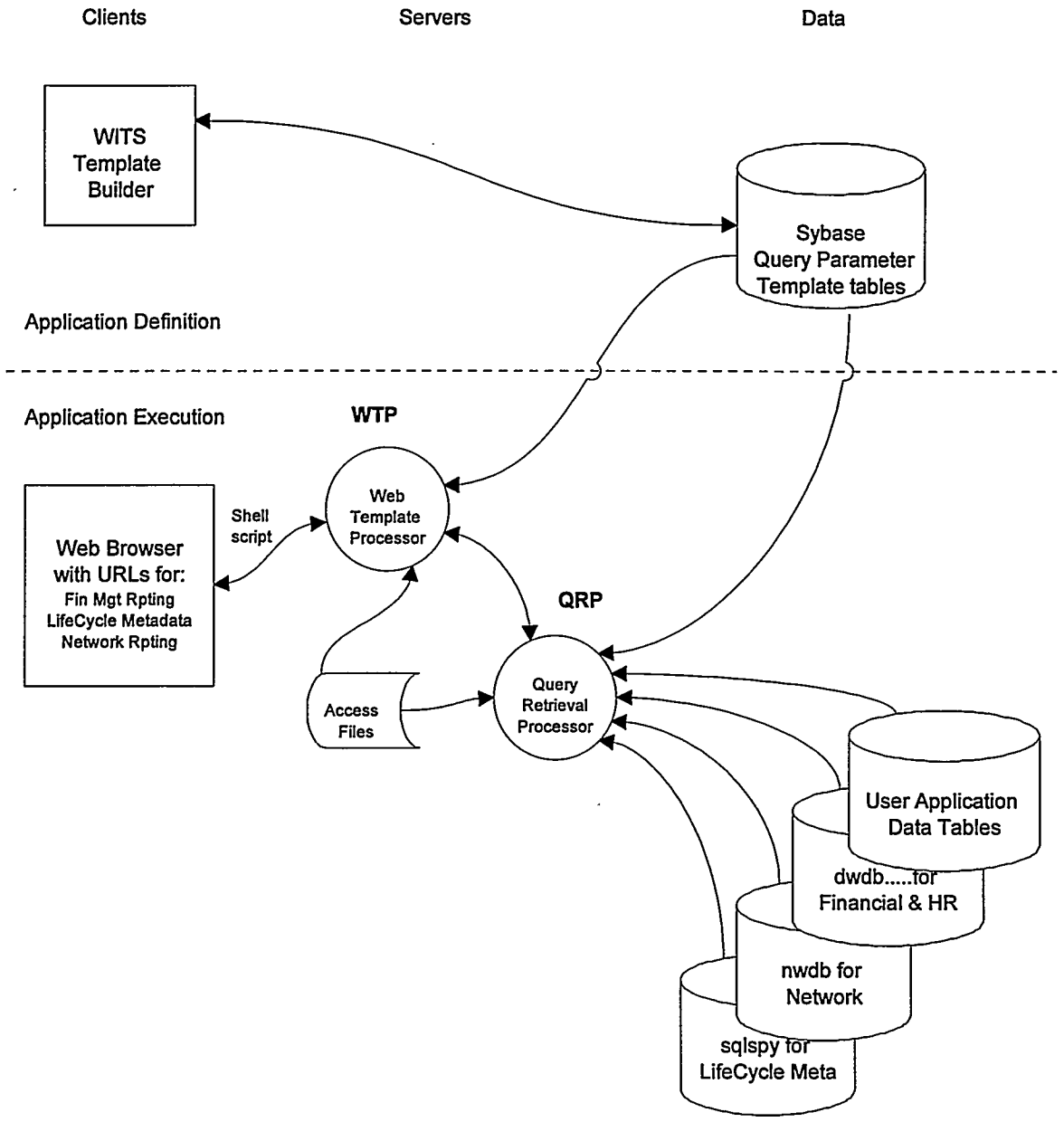


Figure 3.0—1 WITS High Level Functions and Interfaces (Web interface)

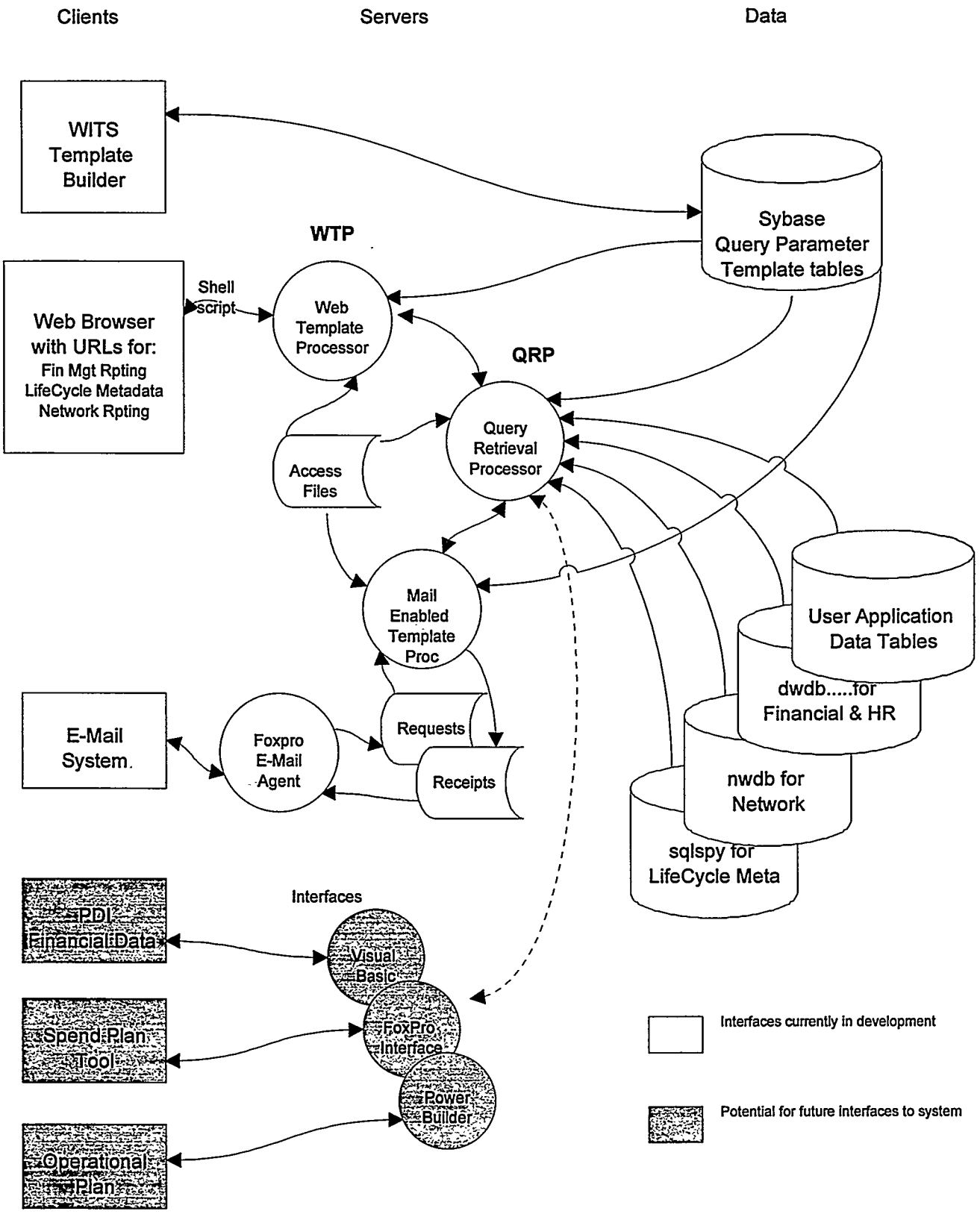


Figure 3.0—2 WITS High Level Functions and Interfaces (additional interfaces)

The following scenario attempts to give the developer a fairly detailed but still overview level of the complete flow of tasks and functions involved in developing and executing an application with the WITS software. This scenario focuses on an application with a Web interface, but much of it is relevant for applications with other presentation interfaces. Much more detailed information about each of these WITS functions and components is available in Section 3.3, Three-Tier Architecture of WITS.

3.1 Overview scenario: Defining the application

Each WITS application is driven by templates and parameters which are stored in Sybase tables. The developer uses a GUI interface called the WITS Template Builder to input and maintain this application definition. The WITS Template Builder insulates the application from the physical data column and function names through aliases. This facilitates the possible conversion from one Relational Database Management System (RDBMS) to another and also allows functions to be inherent in an alias.

A developer will create HTML templates (for Web applications) and data request templates (for all applications) using the WITS Template Builder (see section 4.2, WITS GUI Front END, the WITS Template Builder). The WITS Template Builder is used to establish the relationship between the HTML and the data templates. The Template Builder is also used to establish data access by naming the database, data tables or views, and column names.

The developer also uses the WITS Template Builder to associate each HTML template (Web applications) and data templates (all applications) with the data group that it will access. The data group provides the connectivity requirements to the SQL server.

3.2 Overview scenario: Executing the application

Launching environment:
For a Web application the execution of WITS is initiated by a Web browser. When the customer clicks on hypertext that points to a WITS application, this launches the WITS runtime functions, which return data to the Web Browser. For non-Web applications, a different launching environment may be used.

UNIX Shell script:
The developer also writes a UNIX shell script that will contain the command line required to launch the initial screen of the application. The shell script is referenced via its URL (uniform resource locator) from within the browser, and it then launches the WITS applications. For Web applications, the Web Template Processor (WTP) is launched first, and it launches the Query Retrieval Processor (QRP). The shell script must be located either within or under the Common Gateway Interface (CGI) directory for the Web server where it executes.

Access File:
The UNIX shell script contains the full path and filename of the application's access file (see section 5.4, Access File with Parameters). The access file contains, among other parameters, environment specific URL paths, a broadcast message, and the Database Management System (DBMS) log in and time-out parameters.

WTP:
The WTP function manages the acceptance of query criteria from the web browser, initiates the QRP, accepts the returned data from QRP, and formats it into HTML for the browser environment. WTP will accept from one to many rows of returned data from QRP and itself provides no restrictions to the number of rows that can be returned to the browser.

At first execution, the WTP displays the HTML query page at the browser. If parameters are passed in with the hypertext links, a report can be displayed initially. However, typically there will be no criteria fed in when the HTML query page is first displayed. With the query page displayed, the user can enter the input criteria to the application and submit the screen for processing. Once submitted, the shell script passes the value of the query criteria (stored in an environment variable named QUERY_STRING) as an argument to the WTP. Using the data template, the WTP converts the criteria entered by the user into a command string, which is sent to the WITS QRP.

QRP:
The QRP is the function within WITS that is responsible for performing the actual retrieval of the user data. Although QRP is often accessed by a Web browser passing query parameters through the WTP function, it can also be executed using an interface to other presentation tools such as e-mail or GUIs built by Visual Basic, PowerBuilder, etc.

QRP is a C program that executes in the UNIX environment and uses Sybase System 10's Open Client DB-Library routines for its DBMS access. A properly formatted query command will cause QRP to either dynamically write an SQL statement or else execute a Sybase stored procedure command line to be executed against the DBMS. To avoid the very large number of rows being returned from the QRP, there is a maximum number of rows parameter that QRP uses to manage the limit.

For both Web and non-Web interfaces the QRP uses the template ID to find the data group and the maximum number of rows that can be displayed by this query. The QRP then uses the data group to get the Sybase data base server name, the interfaces file, user ID, and password. The QRP then converts the alias column names (which came from the QUERY_STRING) into the actual DBMS functions and column names.

At this point QRP will build dynamic SQL for a complete SELECT statement, or QRP will execute a stored procedure. For a SELECT, QRP locates the DBMS database name and table name. The QRP converts the command string that it received from the WTP into SQL, and sends this SQL to the DBMS, which retrieves the data from the database and buffers it for the QRP to access.

For Web applications, the QRP sends the set of data back to the Web Template Processor as an array. For non-Web applications, QRP sends the data back to the user's interface as an array. The user interface will format the data as needed.

WTP:
When the WTP receives the array from the QRP, it accesses the templates, which contain the HTML for the Header, Detail, and Footer of the report. The WTP dynamically places the returned data array on an HTML page together with any static text and/or images that the developer included in the HTML templates.

3.3 Overview of WITS Three-Tier Architecture

As figure 3.3—1 shows, the run-time portion of WITS has a three-tier architecture.

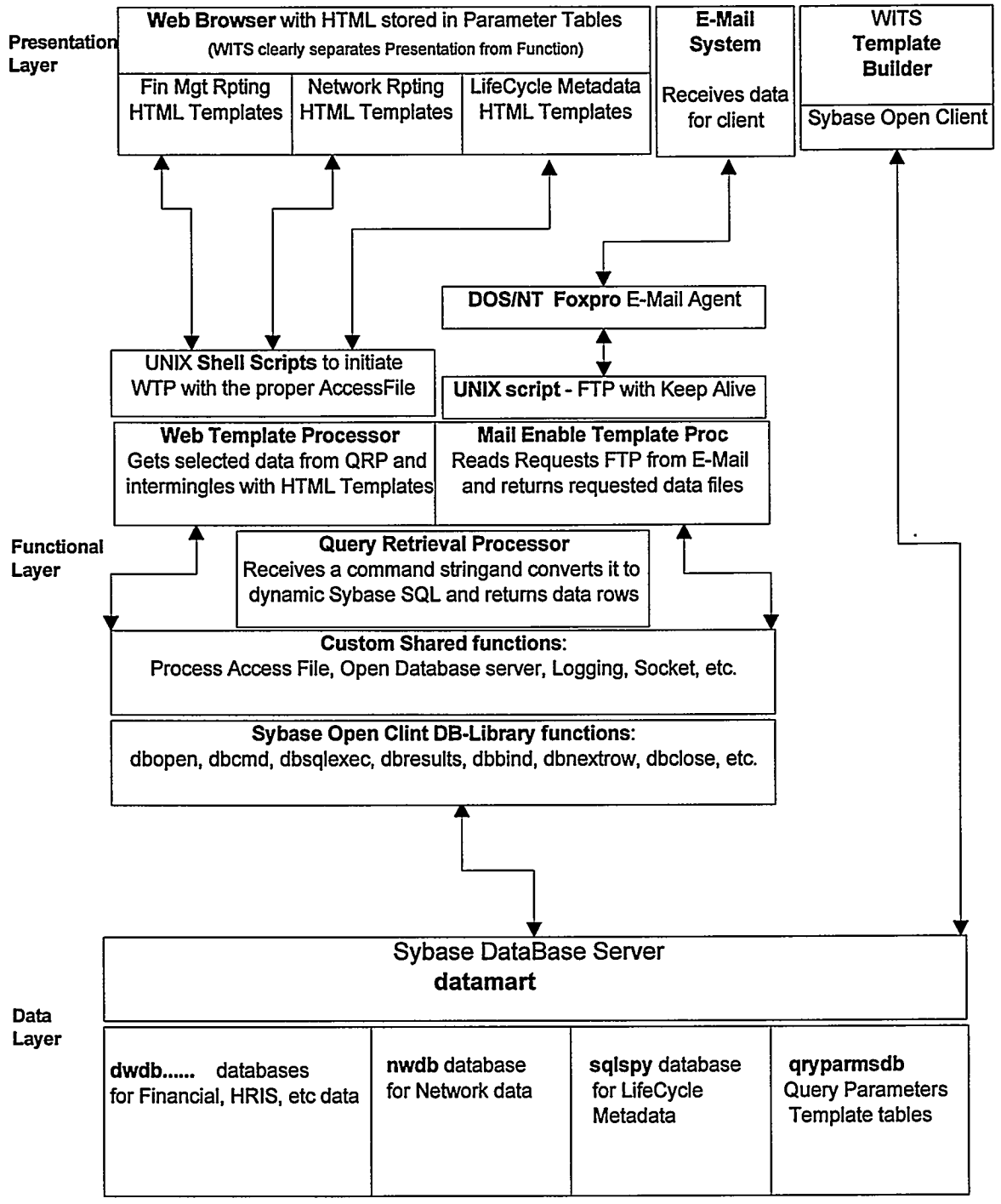


Figure 3.3—1 WITS 3-Tier Architecture

4. Presentation Layer Components

The presentation layer has two separate components. There is the web browser (or other interface), which is a Windows application running on a DOS or NT client machine, and the WITS Template Builder, which is also a Windows application.

The WITS Template Builder is a two-tier client server application development tool. It is used to create the application templates that will be processed by the three-tier run-time portion of the WITS system. The Web browser is the presentation environment for many WITS applications. The HTML source code that defines the characteristics of Web presentation is stored in the data layer and is maintained by the WITS Template Builder.

4.1 HTML Pages with Substitution Tags

HTML pages are defined within the WITS Template Builder as templates. Substitution tags within the HTML templates allow retrieved data to be intermingled with predefined presentation.

4.1.1 HTML Template Pages

There are four components of HTML templates. Q is the Query Screen. H (Header), D (Detail), and F (Footer) are parts of the result screen.

1) **The query screen:** (HTML type "query"). This screen is a complete HTML page definition that allows the user to enter query criteria and submit that criteria to initiate a query request. Although query requests are not restricted to submission from the query screen, this is generally the first screen that is displayed to the user when calling up the application from a Web menu page.

The query screen automatically displays when the shell script is activated without any parameter values except for the Parm template ID. If the developers need to generate a report while bypassing the query screen, then they can do so by passing in parameter values.

Substitution tags allowed in the query screen are for URL path parameters (U1 and/or U2) and the broadcast message parameter (M1).

2) **The results screen header section:** (HTML type "header"). This is the top portion of a complete result screen. All substitution tags can be used in this section, including any column of the resulting data set (Rn).

3) **The results screen detail section**: (HTML type "detail"). This is the middle portion of a complete result screen. All substitution tags can be used in this section. If the resulting data set is for multiple rows, then this section will be displayed repeatedly with substitutions, once for every row of data.

4) **The results screen footer section:** (HTML type "footer"). This is the trailing portion of a complete result screen. All substitution tags can be used in this portion. However, the results tag <Rn> will only allow <R1> here, because the last row returned is a message string that indicates how many rows were returned for the query results.

4.1.2 Substitution Tags

Substitution tags are a method of intermingling the HTML from the presentation with the query results from the data layer. These substitution tags mark a location within an HTML screen where data from a column in the query result set would be inserted. This segregates the presentation layer from the function layer. Substitution tags are also used for moving from one environment to another.

All substitution tags are optional. The developer can be selective about which substitution tags to include on the display screens, or can even omit all substitutions if desired. Omitting substitutions would eliminate the flexibility of dynamically substituting data and URL paths, but WITS allows it.

The format of this WITS specific substitution tag is as follows:

~|%[-][N]s<Rn|Wn|In|Un|Mn|Nn> where:

The ~|% location tag

Means a substitution location has been marked and the actual data must be embedded at this position in the output screen.

The - justification tag

Optional; left justify and pad the output with spaces. If this parameter is omitted, the output will be right justified.

The N pad tag

Optional; pad the output field for N integer number of spaces. If the - is left out then right justification is assumed. This number indicates a minimum field size and cannot be used to truncate the field value.

The s variable type tag

Required; variable type is character. The variable type will always be char because the data is being transmitted through the socket as a character string.

The <Rn|Wn|In|Un|Mn|Nn> substitution type tag

Required; substitution type display tag. Only one of these six possible substitution types is used in a substitution tag. Meanings of each part of the substitution type tag are:

1) **R -** query results column data returned from stored procedures.

2) **W** - where structure criteria values used in dynamic SQL. These W values are stored from the requesting HTML screen's query_string. This can be used to display to the end user of the application the criteria that the user entered to create the result set. The developer does not have to include the criteria in the SQL Select clause. The W values are also available for subsequent query requests from the result screen.

3) **I** - informational structure values. I values are pieces of information that need to be propagated from screen to screen down the hierarchy of an application's screen structure, but are not to be included in the criteria of the query itself. For example, a login screen asks the user for their account name. After submitting the account name, the user's full name and organization is returned. The developer then wants to display the user's full name and organization on all subordinate screens without having to include them in the query criteria on those screens. Using the information tag allows this.

4) **U** - the URL path. U values are the values defined to the access file URLPATH1 & 2 parameters. U values should be placed immediately before the name of either the shell script or other valid file. (i.e. ~|%<U1>goodstuff.script) Notice there is no space between the tag and the name of the script file. This substitution tag is essential for migrating from one environment to another.

5) **M** - broadcast message. The M value is the value defined to the access file BCASTMSG parameter. Typically a developer would use this tag on the first query screen at the top of an application's hierarchy of screens, but it can be used on any section of any screen definition.

6) **N** - non applicable. The N tag is a minor tag that has very limited use. The N tag will cause the characters "N/A" to be substituted on the screen. Its purpose is to allow certain fields to appear on a screen, yet due to the context of the query there would never be any data for the field. Next to the heading N/A is displayed. Doing this with the N tag rather than with literal text next to the heading label text improves formatting consistency at the browser.

The "n" that appears next to the type character is an integer that relates to the array offset position into either the results (R), where (W) or information (I) structures. The integer is either 1 or 2 for the URLPATH substitutions U1 or U2. N and M can have any integer value, but this value is ignored because there is only one value to substitute. This array offset integer is validated against the number of columns returned from the QRP. In the Data Template the numbers representing embedded column data in the HTML results must correspond to the relative column number in the COL definitions of the query request template for SQL statements. The same applies for the WHERE and INFO Data Template definitions.

For example, a substitution tag could be ~|%-12s<R3>. This indicates that the value in the third column of the query result set is to be placed here. The value will be left justified and padded out to twelve characters in length with spaces (unless the value is greater than twelve characters long, then the entire value would be displayed).

4.2 GUI Front End (WITS Template Builder)

The WITS Template Builder provides easy access and setup of the WITS developer's tables. (See section 6.1, WITS Metadata Tables). The WITS Template Builder was written in PowerBuilder version 4.0 and designed to run on a PC Windows platform.

The developer can use the WITS Template Builder to

1. **Create and modify data access for an application**
    - Establish a data group and tell WITS the database your application will access.
    - Set the database access parameters such as database server, userid and password.
    - Create a view alias for each data table/view, to be used in translating from an application's logical names to a database's physical database table/view names.
    - Create an alias for each application logical variable column name, to be used in translating to each database physical column/function name.
2. **Create and modify HTML and data retrieval**
    - Set the template id and associate it with the data group it will need to access.
    - Setup a data template to be used for creating a data request command string.
    - Create templates containing the HTML for each query page and report.

For an overview of how to design the parts of your application that are visible to the user, see section 8, Design the Template Builder parts of your application.

5. Functional Layer Components

The functional layer of WITS is responsible for accepting requests from the browser or GUI environment, translating those requests into SQL server command statements, and preparing the resultant data set for display to the user.

5.1 UNIX Shell Script with $QUERY_STRING and Command Line Parameters

The launching function for any WITS Web application is its UNIX shell script. The UNIX shell script contains a number of Command Line Parameters, and the $QUERY_STRING parameter. The QUERY_STRING is an environment variable known to Web functions (specifically known to the httpd web deamon). The QUERY_STRING is populated with input parameter names and values every time a request is submitted from the browser.

The command line required to invoke WITS for a specific application is located in the UNIX shell script file. The format of the command line with its parameters is:

```
/dir/path/wtp -v -l \
 -a/dir/path/AccessFileName \
 -tDefaultTemplateName \
 -q$QUERY_STRING
```

A sample UNIX shell script containing a command line might appear as:

```
#! /bin/she
/export/home/c/wtp -a/export/home/c/AccessFile -l -tCaseOrgTotals
-q$QUERY_STRING
```

5.1.1 /dir/path/wtp

WTP path - Required for all applications
This parameter contains the path and executable name to the initial function of the runtime portion of WITS. For a Web application, the initial function is the WTP. For a non-Web application, the initial function is whatever function communicates between the user interface and QRP.

5.1.2 -v

Verbose mode - optional for all applications.
This parameter is used to enable and disable the verbose mode for application testing.

5.1.3 -l

Logging mode - optional for all applications.
This parameter is used to enable and disable the transaction logging feature. Putting this parameter on the command line causes each execution (excluding the initial load of an application's query screen) to create a flat file containing the transaction date

and time, followed by the parameters found in its QUERY_STRING. This file will be written into the directory identified by the LOGPATH parameter in the Access File.

An example of the contents of a single log file is:
"Tue Jun 13 15:37:23 1995",
"templateid=loginnamesl&start_date=06/08/95&start_time= 3:04PM".

The purpose of this file is to collect statistics on the frequency and volume of application templates being viewed. The value following the "templateid=" parameter indicates the application template the transaction belongs to. This file was formatted with the two strings (transaction date, and QUERY_STRING value) enclosed in double quotations and comma delimited for easy import capability into a database for reporting purposes.

5.1.4 -a/dir/path/AccessFileName

Access file location - required for all applications.
This parameter identifies the directory path and filename of the Access Parameter file defined in section 2.7.2.

5.1.5 -tDefaultTemplateName

Default Application template - required for Web applications using WTP.
Although each application only requires a single shell script to launch from, it may consist of many template definitions, each of which provides a screen display of retrieved data.

At the initial load of the application, the QUERY_STRING will always be empty and will not contain the "templateid=" parameter which specifies the template screen to use. At this initial load the WTP process will use the -t template name to retrieve the template's query screen and display it back to the browser for a query request.

On subsequent executions of the UNIX shell script, the QUERY_STRING will not be empty because it can be dynamically populated through the HTML screen. See the following section.

5.1.6 -q$QUERY_STRING parameter

QUERY_STRING environment variable - required for Web applications.
This is the only parameter in the UNIX shell script that is not a command line parameter. The query string is composed of HTML input parameters and their values at submission time. It is used to pass the value of the Web server's Common Gateway Interface environment variable into the WITS WTP. This must occur every time a request is submitted from the browser.

The format of the QUERY_STRING is as follows: "parmid1=value& parmid2=value &...". The parmid's in the QUERY_STRING relate directly to a value in the has_parm_value column of the data template (as defined in the WITS Template Builder). These parameters are the aliases that will be used for data access They provide the insulation of the application from the physical database, and the ability to use database functions transparently.

5.2 Web Template Processor (WTP)

The WTP function is called directly from the UNIX shell script and serves as the translator from the HTML query request to the command syntax that the WITS QRP requires.

The purpose of WTP is to accept a query request from the browser and to format the query criteria, based on the template definitions, into a query command that the QRP will understand. WTP establishes the communications needed to converse with the QRP, sends the query command to QRP, and receives the resultant data set or error response code. After receiving the resultant data WTP formats the data (see section 4.1, HTML) into HTML syntax and displays the results back to the browser. See Figure 5.2—1 Context Diagram for the Web Template Processor.

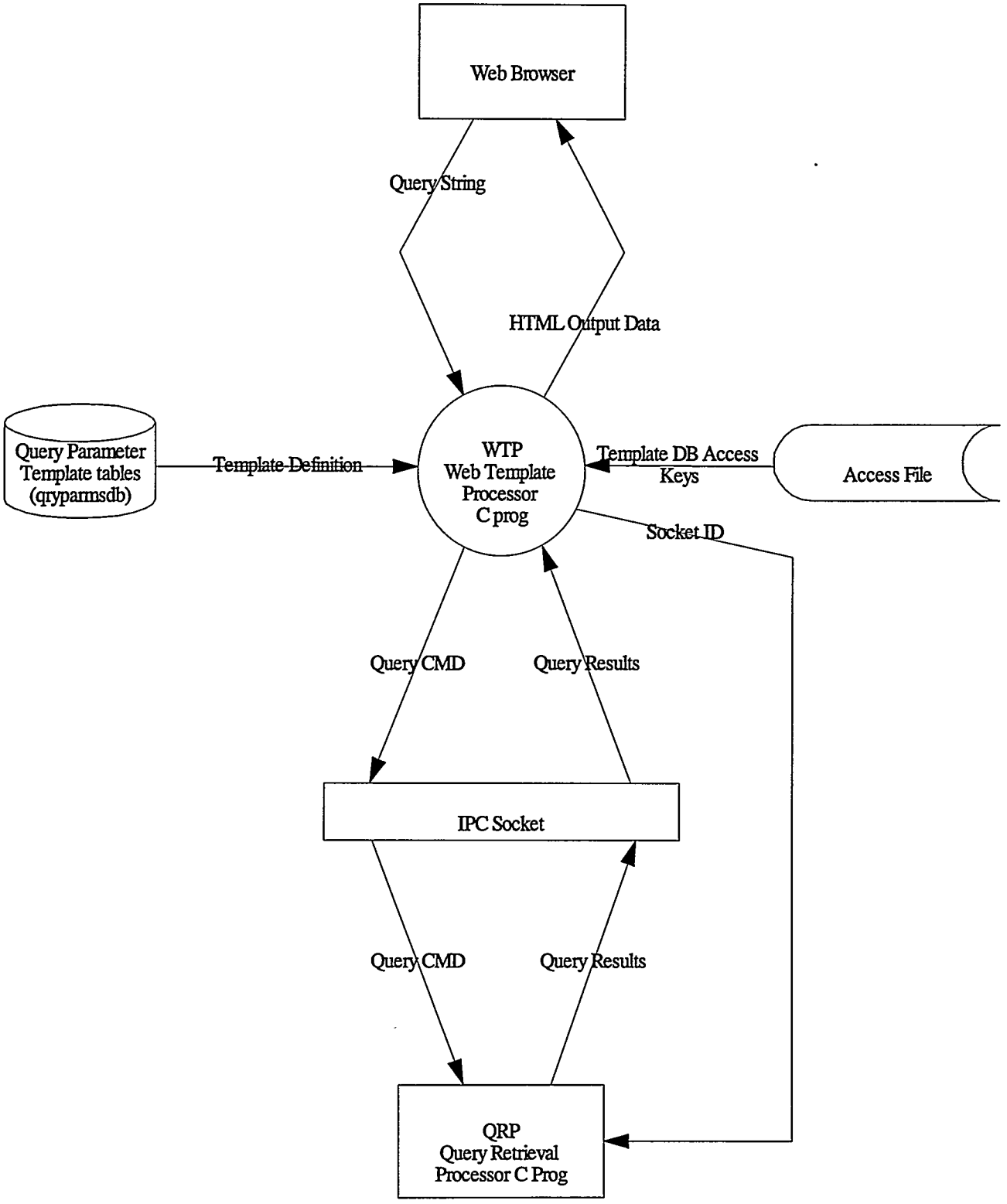


Figure 5.2—1 Context Diagram for the Web Template Processor

5.3 Query Retrieval Processor (QRP)

The WITS data retrieval function, the QRP, was designed to flexibly support reusability by many different systems. It provides consistency in definition of functionality across applications.

The QRP uses the column_function_alias and view_location tables to provide a translation from the query commands' alias columns to the actual Sybase table and column/function names. That translation keeps these application processes insulated from any structural or naming changes that the DBA staff may make to the database. The translation allows database functions to be called transparently by using an alias name. It also allows the application to dynamically choose, at run time, the column to be retrieved.

The purpose of the QRP is to accept a query command via an IPC socket interface, convert the command components into either a valid SQL statement or stored procedure exec statement, and to execute that against the SQL server. All rows retrieved from the SQL server are formatted into a delimited string array and returned to the requesting process via the socket. The array has no column headings, and each column is separated by a pipe symbol. The array is in the format "Column1Value|Column2Value|Column3Value|... (CR,LF)", where CR,LF means Carriage Return, Line Feed. See
Figure , .Context Diagram of the QRP function.

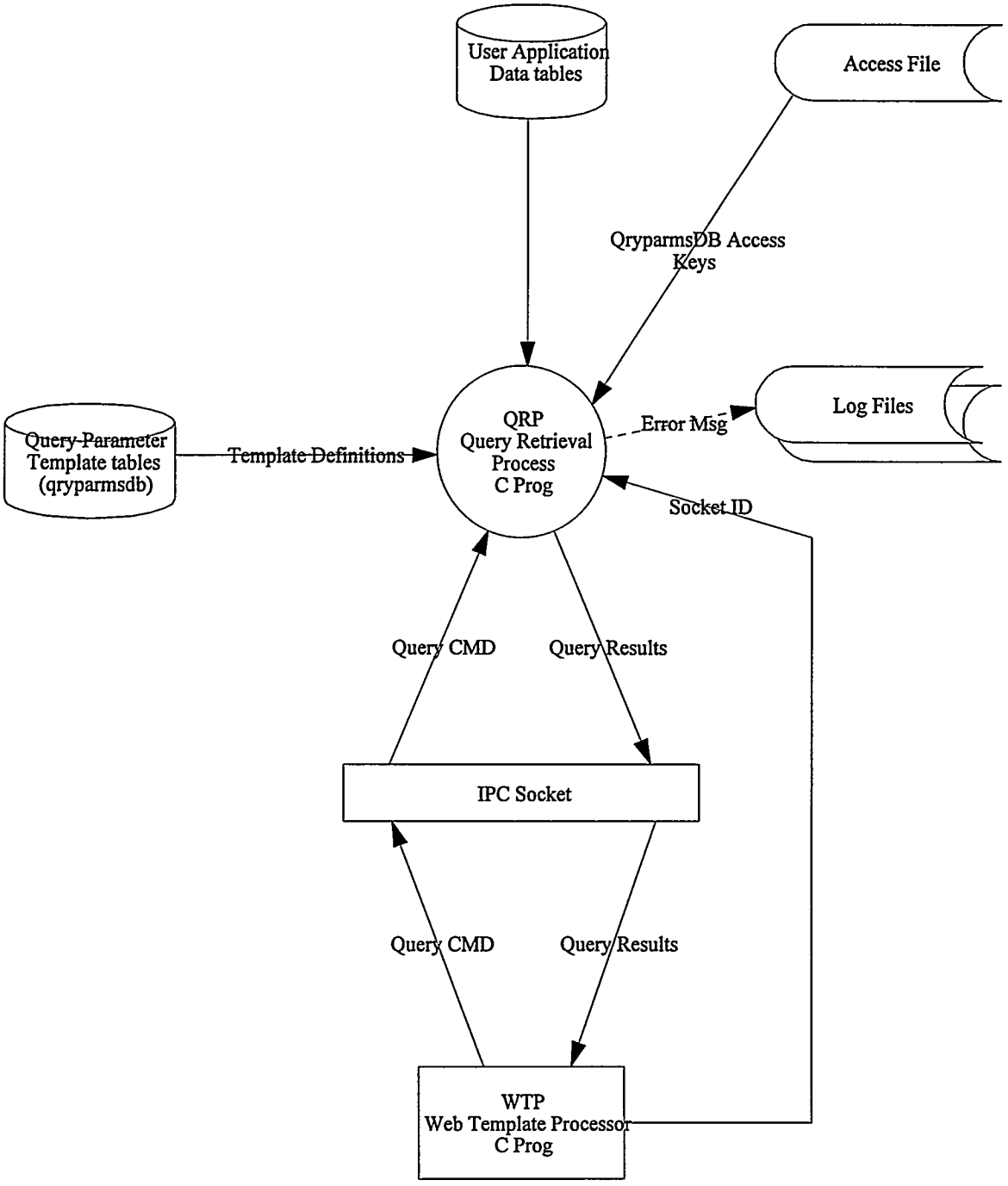


Figure 5.3—1 Context Diagram for the Query Retrieval Processor

5.4 Access File with Parameters

The Access File contains parameters specific to the environment, database, and broadcast messages. The full directory and filename path to the access file is specified in the UNIX shell script.

The order of the parameters within the access file does not matter; however, each parameter must begin in column one. Disabling an optional parameter can be done simply by shifting the parameter to the right so that it does not begin in column one. Parameter names are case sensitive and must be upper case.

Additional lines other than the defined parameters, such as comment lines, can appear in the access file. No special comment character is needed; simply begin each comment line after column one.

Note that unpredictable results may occur if anything appears after the parameter value on the same line. There is no way to append a comment to the end of a parameter. Be sure that a space or spaces do not follow the end of the value on a parameter line. This will also result in unpredictable results.

Following is an explanation of the definition and meaning of each of these parameters:

5.4.1 INTERFACES=/dir/path/to/Sybase/interfacesfile

Required for all applications.
The interfaces parameter in the access file provides the full path and filename to the Sybase interfaces file. Each server that hosts the WITS executables must have its own interface file. This interface file contains the IP address of the host machine where the database server is located.

There are other ways to provide the Sybase Open Client system with the IP address of the machine hosting the database server. However, passing the explicit location of the interfaces file allows multiple interface file definitions to be used. This allows the developer to move the application from one server to another and simply change the access file to point to the appropriate interface file.

5.4.2 SERVER=sybgila

Required for all applications.
This parameter in the access file contains the name of the SQL server instance where the target database is located. The SQL server name on this parameter must be in the interfaces file.

5.4.3 DBNAME=qryparmsdb

Required for all applications.

This parameter contains the name of the database within the SQL server instance which is declared on the SERVER parameter. This identifies the database where the WITS application templates have been defined.

5.4.4 APPNAME=APPL1

Optional for all applications
This parameter is a name specific to the application group that will be used when WITS logs in to the SQL server. Although this parameter is optional it is strongly recommended. It provides the DBA staff a means of determining which WITS applications are running against the SQL server. Limit the name entered here to no more than 26 characters in length. Sybase application names can be up to 30 characters in length, but WITS automatically prefixes 4 additional characters to whatever name is entered here (i.e., WTP-APPL1 or QRP-APPL1).

5.4.5 UID=userid

Required for all applications.
This parameter is the user account ID that the WITS application will use to log in to the SQL server. For security purposes this userid should be given select authorization only (a read-only account).

5.4.6 PWD=password

Required for all applications.
This parameter is the user account password that the WITS application will use to log in to the SQL server.

5.4.7 SQLTIMEOUT=60

Required for all applications.
This parameter is the SQL server command execution time-out parameter, in seconds.

5.4.8 LOGINTIMEOUT=3

Required for all applications.
This parameter is the SQL server login time out parameter, in seconds.

5.4.9 DATASERVERPROG=/dir/path/to/qrp

Required for Web applications using WTP.
This is the full directory path and filename to the location of the WITS QRP. The actual program name is case sensitive and is lower case; "qrp".

5.4.10 SOCKETPATH=/dir/path/to/sockets/

Required for all applications.
The is the full directory path to the location where the UNIX IPC socket handles are to be written. Sockets are the method through which the two processes WTP and QRP communicate. The UNIX directory declared here must allow write permission to the UNIX account that will be executing the WTP and QRP processes. For

example, in the browser environment an account has been assigned to the httpd deamon for all web activity. The permissions for that account must be read and write enabled to the socket directory, or the WTP and QRP processes will not be able to communicate.

5.4.11 URLPATH1=http://devlab-sun/cgi-bin/dev/

Required for Web applications.
This is the URL for the web browser environment specific to an application. This value is used during the substitution cycle of the WTP process. The purpose of the parameter is to provide flexibility when migrating an application from one environment to another, such as from development to testbed to production. Because all HTML is maintained and run from a Sybase database it can be awkward to embed a URL in the screen definitions. The URLPATH1 parameter eliminates the need to edit the HTML definition after the application has been approved and migrated to its next stage of the deployment cycle.

URLPATH1's intended use is to declare the full path to the directory where the shell scripts are located. For details on the UNIX shell script, see section 5.1, UNIX Shell Script with $QUERY_STRING and Command Line Parameters.

URLPATH1 can also be used to dynamically change the path within hypertext links from one HTML screen to another.

5.4.12 URLPATH2=http://devlab-sun/fisdata/

Required for Web applications.
URLPATH2's purpose is identical to URLPATH1 with the exception that it gives an alternate location for files to be referenced. Typically this is used to reference anything other than the UNIX shell script that invokes the run time environment. These files might be HTML source files for help, or multimedia files such as images, movies, or sound.

Both URLPATH1 and URLPATH2 must be defined in the access file. There are no rules requiring that these paths be used in the HTML substitution process.

5.4.13 LOGPATH=/dir/path/to/log/files/

Required for all applications.
This is the full path to the directory where the .sql and .log files are to be written. If an unexpected DBMS error occurs, a log file will be created containing the time and date, along with the DBMS generated error message. The log file will be written to this location. If the logging option has been enabled, (see 5.1.3, the -l parameter of the UNIX Shell Script) then .sql files containing the format specified in that section will be written to this location.

5.4.14 BCASTMSG=<blink>This is a broadcast message that blinks</blink>

Optional for Web applications.

This is the application Broadcast message. This optional parameter can be used with any substitution tag, HTML formatting tag, or hypertext link on any query screen to communicate a message that would be relevant to the end users. For example, it could be used to notify the users of enhancements, planned system down time, etc. The message is limited to 199 characters in length. Any characters beyond that will be truncated.

5.4.15 PRINT$=1

Optional for Web applications.

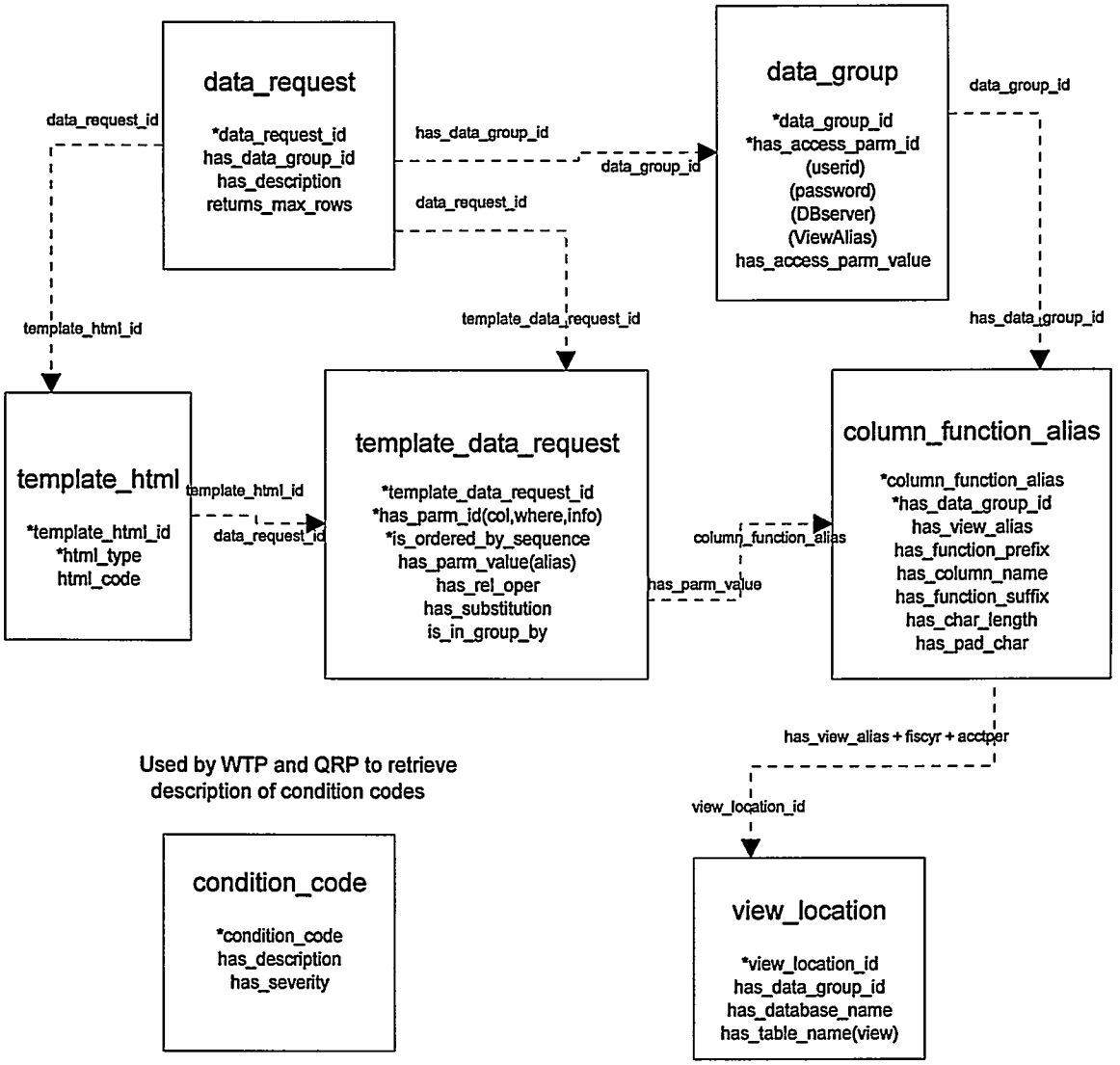
The PRINT$ parameter provides special formatting for money data fields. There are four options to this parameter. They are listed here with bullets rather than numbers to avoid confusion with their 0-3 values.

- PRINT$=0 Print money fields with commas only. This is the default state for money fields.
- PRINT$=1 Print money fields with commas and a dollar sign only on the first row returned in the set.
- PRINT$=2 Print money fields with commas and dollar signs on every row in the set.
- PRINT$=3 No formatting other than a decimal.

6. Data Layer Components

6.1 WITS Metadata Tables

Figure shows tables, column names, keys, and relationships between the metadata tables.



* indicates the unique key of the tables

Figure 6.1—1 WITS Query Parameter Template Tables

The tables containing information needed for WITS to function are maintained in a Sybase database. These tables are listed below in alphabetical order. Detailed information on each data field is available in the WITS Microsoft Windows Help file that comes with the WITS Template Builder.

6.1.1 column_function_alias Table

The column_function_alias table is used in the translation from the logical HTML or other interface names to the physical database. It is used by the WITS QRP to translate each column/function alias in the template_data_request table to the corresponding DBMS column name or function of a column name.

The column_function_alias table is central to much of the functionality and benefits provided by the QRP. For example, the QRP uses the column_function_alias and view_location tables to provide a translation from the application's query commands to the creation and execution of dynamic SQL, and from the application's PROCs and WHEREs to the execution of stored procedures. The translation from application parameters to physical database column/function names keeps these application processes insulated from any structural or naming changes that the DBA staff may make to the database. The translation allows database functions to be called transparently by using an alias name. It also allows the application to dynamically choose, at run time, the column to be retrieved, based on user input.

The column_function_alias table has a column that is also called column_function_alias. This column associates an alias for a database column name, or function of a data column, and also identifies its view_location_id. The view_location_id can then be used to access the view_location table, thus finding the corresponding database name and table/view name where the data can be accessed.

The column_function_alias is the same as the eighty-character has_parm_value column of the template_data_request table. This alias must be used to name variables in the HTML pages or other user interface to identify the data or function name. It is used in the translation from the logical HTML or other interface names to the physical database names.

By looking at the column_function_alias table on the WITS review screen, you can identify all the DBMS data columns/functions that are available to be used in a data request for an application.

Each column_function_alias belongs to a data_group_id. They are associated because the has_data_group_id column of the column_function_alias table holds the same value as the data_group_id column of the data_group table.

Key Field:

has_data_group_id
column_function_alias

Other Fields:

has_char_length
has_column_name
has_function_prefix
has_function_suffix
has_pad_char
has_view_alias_id

The column_function_alias table is populated by using the Column Alias window. For details of how to do this task, see Section 10.5.2, Establish a column alias.

6.1.2 condition_code Table

The condition_code table allows the WITS developer to store the text of all error messages, and to identify each message with a unique condition code ID. The application developer does not need to deal directly with the condition_code table, although it may be useful to recall the text of WITS messages during testing of the developer's application.

The WITS database comes with a predefined set of message descriptions; however these messages can be customized if needed using the WITS Template Builder. The has_severity field is not currently used in WITS, although it may be used in a future enhancement.

Key Field:

condition_code

Other Fields:

has_description
has_severity

6.1.3 data_group Table

The data_group table allows you to establish a data group id and populate all the access parameters needed to get to that data group. The data_group table is used by the WITS QRP. It provides the access parameters (userid, password, interface paths, and database server) that are needed to access the data. It also determines the database name if a stored procedure is to be used. The long term purpose of the data_group table is to support multiple concurrent DBMS process sessions.

Each user screen that is an HTML template (uniquely identified by a template_html_id) is associated with a particular data_group_id in the data_group table. The data_group_id controls the access parameters for a set of queries. The data group identifier can be associated with more than one template_id.

Key Fields:
data_group_id
has_access_parm_id

Other Fields:
has_access_parm_value

A new data_group table is created on the Insert a New Data Group window. The access parameters for a data group can be changed on the Access Parameters window. For details of how to do these tasks, see section 10.4, Create, Change, and Delete Data Groups.

Sybase access parameters in the data_group table also occur in the Access File. They are cross referenced to their access file counterparts as follows:

Access_parm_id		**Access file**
DBNAME	=	DBNAME
DBSERVER	=	SERVER
INTERFACES	=	INTERFACES
PASSWORD	=	PWD
USER ID	=	UID

6.1.4 data_request Table

The data_request table allows the WITS QRP to find out what database to access in order to return the data to the HTML page that the end user is viewing. It also sets the maximum number of rows that can be retrieved from a query.

Key Field:
data_request_id

Other Fields:
has_data_group_id
has_description
returns_maximum_rows

The data_request table is populated on the Define Template ID window. For details on how to do this task, see section 10.6, Define Template IDs.

6.1.5 template_data_request Table

The template_data_request table is used by QRP to convert the query string (which came from the HTML page through the Shell Script or was hardcoded in the URL) into a command string for the Data Server. It can also be used by non-Web applications to identify variables. The sequence number controls the order by which the querystring is converted.

The has_parm_value field contains the application alias for a column (or function of a column) in a database table. For dynamic SQL, this value will match the column_function_alias field in the column_function_alias table, allowing translation to the real database column/function name. Any parameters that are passed into a stored procedure must be in the has_parm_value field. This value will match the column_function_alias field, and is used to determine the data type (i.e. character or numeric).

The has_parm_id field appears in the Template Builder as either COLUMN, INFO, PROC, or WHERE, however these values are stored in the database as COL, INFO, PROC, or WHERE. The value indicates the type of data access to be used. COL indicates dynamic SQL. PROC indicates a database stored procedure, and there will be no COLUMN fields. INFO indicates that information is not being retrieved, but is simply being passed from one HTML page to another. WHERE indicates the selection criteria that will be obtained from the query string or other interface.

In Web applications the values for the template_data_request (COLUMN, INFO, and WHERE) can be displayed on the HTML page, using the substitution tags in the template_html table. In that table <R>corresponds to COLUMN results, <I> corresponds to INFO, and <W> corresponds to WHERE. For stored procedures, <R> corresponds to the columns returned after execution of the stored procedure. See section 4.1.2, Substitution Tags.

The has_rel_oper field contains a relational operator and only applies to parm_id's of type "WHERE". At this time the list of available relational operators is limited to LIKE and =.

The has_substitution field allows the end user to dynamically select a column (or function of a column) at the input screen. It is a Boolean field and only applies to parm_id's of type "COLUMN". It is used to indicate that the value in the has_parm_value field for that "COLUMN" type is not a column alias name to be located in the column_function_alias table. Instead it is the variable containing the column name selected on the HTML query page, and passed in through the QUERY_STRING for that has_parm_value field.

The is_in_group_by field is a Boolean field and only applies to parm_id's of type COLUMN. This field tells the SQL statement construction to include this column, (once it is translated to the actual column name) in a group-by clause.

The is_ordered_by_sequence field is an integer value that controls the sequence for the order of the columns in the SQL statement. It is this number that is referenced when setting up the substitution tags for the placement of the query result columns in the output HTML screen.

Key Fields:

- template_data_request_id
- has_parm_id (COLUMN, INFO, WHERE, PROC)
- is_ordered_by_sequence

Other Fields:
- has_parm_value
- has_rel_oper
- has_substitution
- is_in_group_by

The template_data_request table is populated on the Data Template window. For details of how to do this task, see section 10.7.2, Define Data Templates.

6.1.6 template_html Table

The template_html table contains the HTML code for the query page and report layout. It is a Sybase repository of the HTML query screens and reports. The template_html table is used by the WITS WTP to display the results data on the Web browser. Each HTML report page requires a template header, detail, and footer. At least one record of type query is required for all templates within a group_id.

Key Fields:
- template_html_id
- has_html_type (Q,H,D,F)

Other Fields:
- has_html_code

The template_html table is populated on the HTML Template window. For details of how to do this task, see section 10.7.1, Define HTML Templates.

6.1.7 view_location Table

The view_location table allows you to create a view alias and associate it with a database and data table or view. The WITS QRP uses the view_location table during the creation of its dynamic SQL to take a table/view alias and find the real database name and table/view name.

Key Field:
- view_location_id

Other Fields:
- has_data_group_id
- has_database_name
- has_table_name

When WITS was used to create the Financial Management Reporting application, code was written to specifically look for and concatenate the acctperiod (Accounting Period) and fiscyr (Fiscal Year) together to form the true database name. This was because financial data is stored in multiple databases, one database for each accounting period. The view_location_id

for financial data consists of the view_alias, the fiscal year, and the accounting period. All financial data requests had to include fiscal year and accounting period criteria, or the system would default to the current month.

The view_location table is populated on the Table/View Alias window. For details on how to do this task, see section 10.5.1, Establish a Table/View Alias.

6.2 User Application Data Tables

These are the tables needed by the developer to hold data particular to the user application. At this time the user application data tables must be Sybase tables. Other relational databases could be used in the future.

7. How to Obtain and Install the WITS System

7.1 Prerequisites - Operating Environment

The GUI interface (WITS Template Builder) was developed in PowerBuilder version 4.0 and runs on a PC Windows platform. It uses Sybase Open Client Client-Library.

To run the Template Builder you will need Windows, the WITS Template Builder executable file and related files, and Sybase Open Client Client-Library (SQL10). You will not need PowerBuilder.

7.2 Prerequisites - Data Base Environment

The WITS template tables are defined and administered by the corporate database administrators. The tables are located in the central Sybase DBMS.

To get update access to these tables, a developer must be defined as a Sybase user. Contact the Central Computing Help Desk (CCHD), which will forward your request to a DataBase Administrator (DBA) The DBA will then categorize your application with a data group ID, define views for your data group, and grant update authorization to requested userids.

If you have a DBMS on a local server, you can get Data Definition Language (DDL) to define your own database. Contact CCHD, who will forward your request for these DDLs to the D.B.A. group.

7.3 Prerequisites - Developer Knowledge and Skills

To use WITS in their software projects, developers must have a knowledge of HTML, basic UNIX, and some relational data retrieval expertise or access to a DBA.

7.4 Download the WITS GUI Front End and related files

The WITS GUI front-end software and on-line documentation (developed for execution on a PC Windows platform) are available for distribution.

1) In your browser, navigate to http://www-irn.sandia.gov/wits/witshome.html, the WITS home page.
2) Follow the instructions on the Web page.

If you are not connected to the Web, call the SNL CCHD for information on how to get connected.

7.5 Install the WITS GUI Front End (WITS Template Builder)

Instructions for installing the Template Builder are available at the Web location shown above.

8. Design the Template Builder parts of your application

You, the developer, will do most of the tasks involved in building a WITS software product inside the GUI interface, the WITS Template Builder.

8.1 Dependencies - Defining a WITS Application

When defining a WITS application, certain tasks are dependent on others as shpown in the following chart. The numbers in the task boxes show where to find text explaining the task.

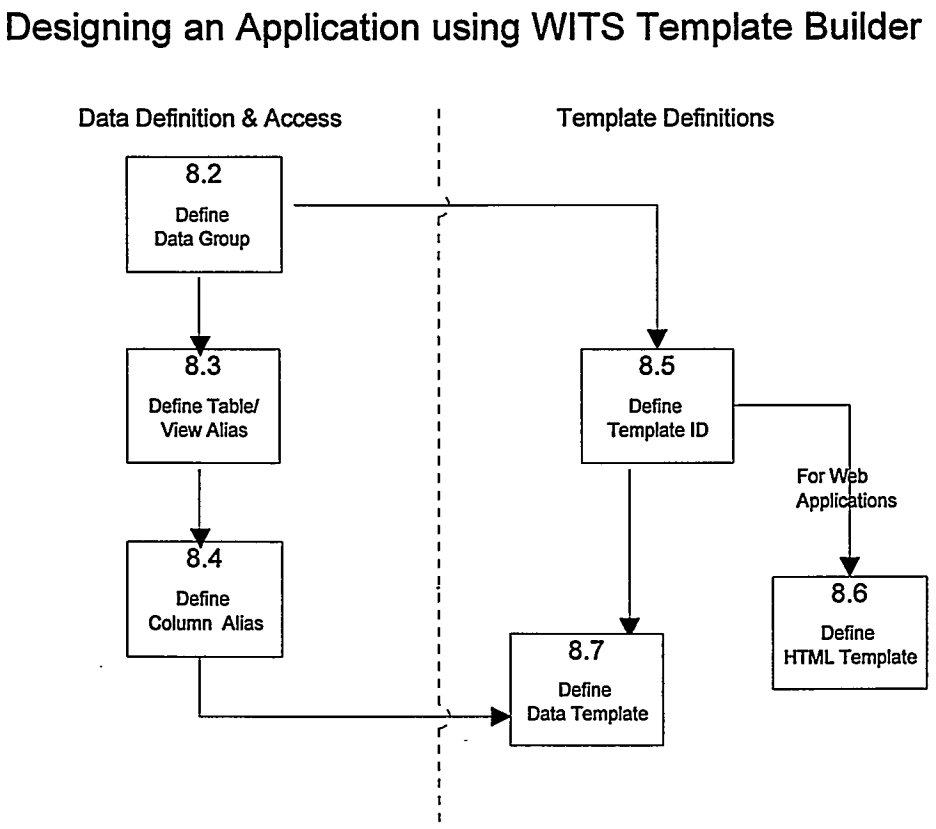


Figure 8.1—1 Dependencies - Defining a WITS Application

8.2 Identify the data required and define by data groups

Identify the type of data you plan to use in your display screens. Much of the data in the Sybase data warehouse will already be defined in WITS. You can review the data groups and columns that are already defined in WITS. If the data you need to access is already defined, you can skip to the authorization step in section 8.2.2, Get Authorization to update views for each data group.

8.2.1 Define new Data Group

A data group generally identifies an application area where the data is created and maintained. It can include either a whole database or one or more tables within a database. If the data you need to access is defined to the DBMS but not to WITS, you will need to define a new data group. This will include supplying the database name, the DBMS server where it resides, the userid and password for query access. For details of how to actually define data groups in the Template Builder, see section 10.4, Create, Change, and Delete Data Groups. If your data is not defined to the DBMS, see section 7.2, Prerequisistes - Data Base Environment.

In a later design step you will decide which table/views are associated with the data group, then you will specify the data columns.

8.2.2 Get Authorization to update views for each data group

The WITS template tables can be used by multiple applications, so it is important to restrict update access to these tables. This is done by defining a set of views for each data group with a limited set of users given update access to those views. These views are also used to manage the migration environment.

An advantageous strategy in developing new applications is to copy and paste from existing definitions, then make modifications as needed. It is possible to see all the data in WITS tables, however you can only update data that relates to a data group where you have update authorization. This eliminates the possibility of developers inadvertently modifying each other's definitions.

If you have not yet done so, you will need to communicate with the Database Administration group to get update authorization to the template tables in the WITS Template Builder for each data group you need to maintain. See section 7.2, Prerequisistes - Data Base Environment.

8.3 Define Table/View Aliases

Once you have defined your data group, you need to establish aliases for all the required tables/views and data columns. These aliases will point to the actual DBMS names.

Each table or view should have a unique alias of five or less characters. Enter this alias along with the actual DBMS table/view name and database name in the Table/View locations table. When the QRP builds the dynamic SQL, the column names will be prefixed with the

table/view alias. This allows for table joins a feature that will appear as an enhancement in a later version of WITS.

For details on how to define the table/view locations, see section 10.5.1, Establish a Table/View Alias.

8.4 Define Column Aliases

Each data field that you want to access via dynamic SQL must be assigned a column function alias in the column alias table. You can also specify different functions of the same physical data column by giving each function a unique column alias when you enter a function prefix and suffix for that data column.

This allows you to present the data differently using native functions of the DBMS (substrings, sums, averages, conversions, upper, etc.). Each record of the column alias table represents a unique way to access the data column and is uniquely identified by the data group and column alias name. Later you will need to enter the column alias name (the same as the HTML parameter name) in the template data request.

The Column Alias table also allows the user to choose the DBMS column name from a drop down list box and have the column included in the dynamic SQL statement.

For stored procedures, instead of creating a record in the column alias table for the data fields displayed, create a record for only those parameters that will be passed to the stored procedure.

For details of how to actually perform this task in the Template Builder, see section 10.5.2, Establish a column alias.

8.5 Design the Screen Navigation and define Template Ids

The word "screen" is used often in the following discussion because it is a generic term that is familiar to software developers on all platforms. Be aware that when talking about WITS applications, the term "screen" is synonymous with the term "template". Screen, screen template, and template all have the same meaning in this context. What shows on a user's screen was previously defined as a template inside WITS.

8.5.1 Design screens and the navigation between screens

Determine each unique screen layout along with the data it will contain and the criteria for selecting that data. The goal is to make as few screen templates as possible and reuse the same templates whenever possible. Screen formats are typically either a query screen where the user enters selection criteria, a listing of data rows, detail of one of the data rows, or detail of a particular data field. Note that designing the navigation between screens is a manual process that is done outside the WITS Template Builder.

The entry point to an application is normally a query screen, which could contain a drop down box of all available reports that use the same query criteria (rather than making multiple query screens). If you are developing multiple query screens, your first screen template might be a menu with links to each query.

In HTML it is possible to navigate from screen to screen by embedding variable data fields in hyperlinks. Review the data required on each screen and identify the possible navigation between screens (HTML templates). For a model Template Navigation Diagram, see Figure 8.5—1 HTML Screen Navigation. You may develop more than one path to each screen, that is, your end user can hyperlink to the same screen template from different places. Each screen template can access data from only one data group. If your end users will need to access related data in another data group you can develop a hyperlink to go to the other data group's screen template for that data.

Figure 8.5-1 shows how to use hyperlinks to drill down to detail data or jump to related data.

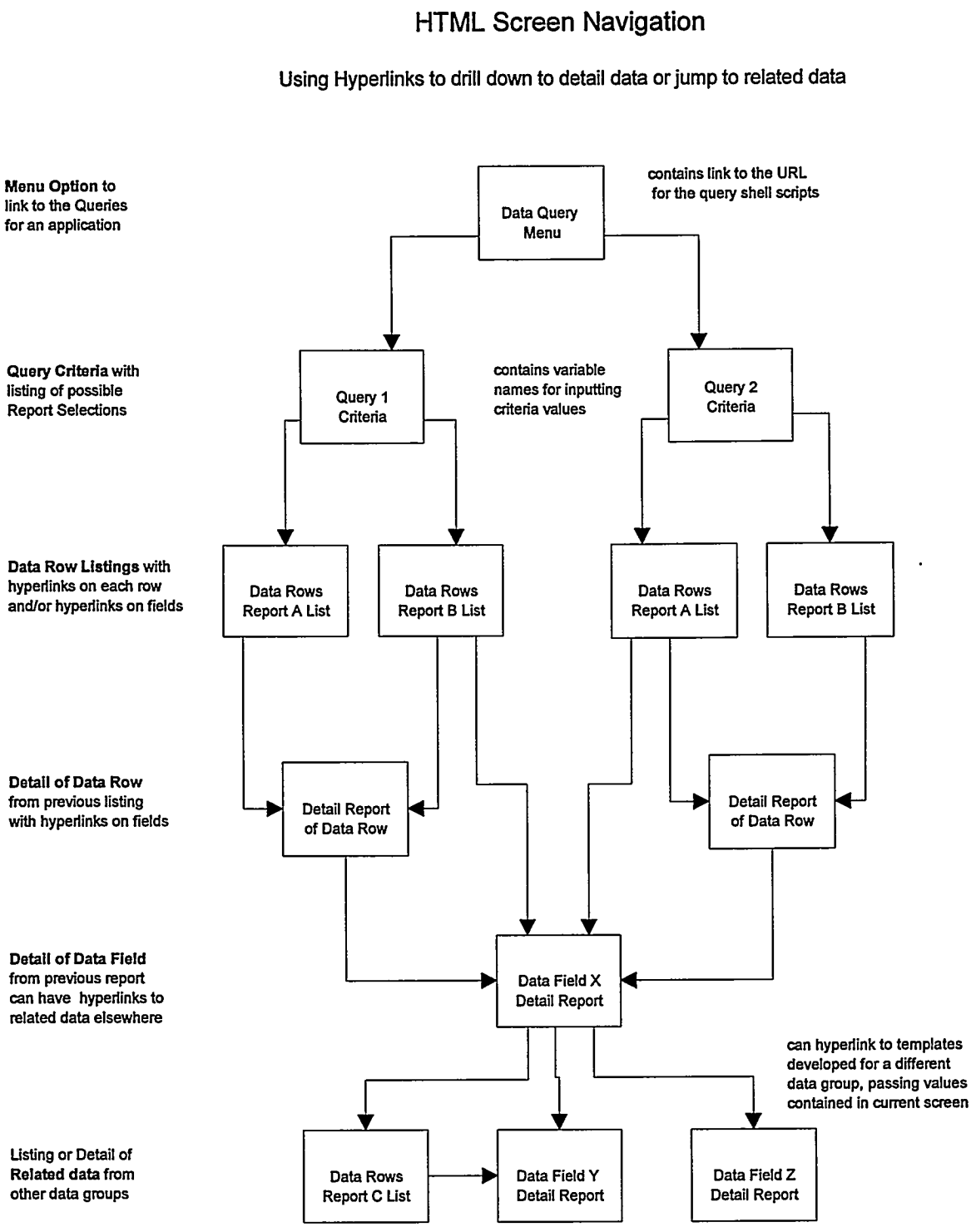


Figure 8.5—1 HTML Screen Navigation

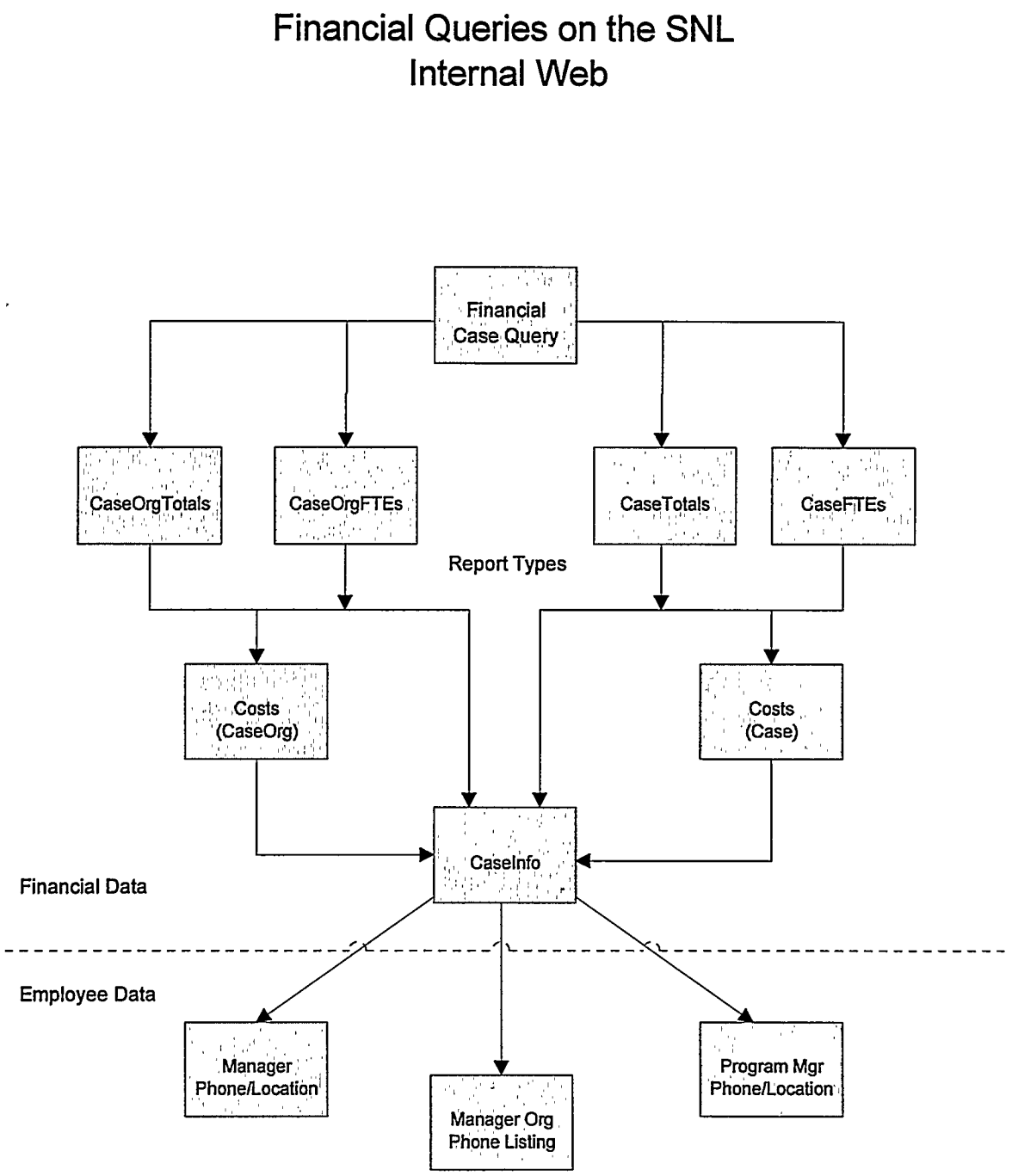


Figure 8.5—2 Screen Navigation, Financial Case Query

Figure 8.5—2 Screen Navigation, Financial Case Query, diagrams the screen navigation used for one of the Financial Queries on Sandia's Internal Web. The Financial Case Query uses query criteria to select data and optionally produce multiple report formats.

8.5.2 Define a Template ID for each screen

Once you, the developer, have identified all screen layouts for the data group, you will need to,assign a template ID to each screen. This will include assigning a unique template id, a description, and specifying the maximum number of rows you wish to display for the end user.

Specifying the maximum number of rows to display prevents the Web browser from grabbing all the end user's memory or disk space on queries which return very large volumes of data. The end user can change the query criteria to a higher level of summarization or a narrower selection if their initial query exceeds the limit.

For details on how to actually define Template IDs, see 10.6, Define Template IDs (Populate the data_request table).

8.6 Define the HTML templates

The html_template table has four types of templates, header, detail, and footer, for a results screen, and query for a query screen.

8.6.1 Define HTML Templates for Query Screens

You, the developer, have the option of creating a query screen to accept input criteria. The query screen is by definition static HTML, it is not populated dynamically with data. If you wish to present a screen already populated with some data, then have the end user input drive a query, that screen must be defined as a result screen, not a query screen. A query screen (static HTML) is presented to the end user as a result of one of the following scenarios, not as a result of SQL running:

- a shell script passing an HTML template name as the template-id and no other parameter values
- a link in a HTML template, which contains the template id and no other parameter values

The query screen will be the top level of the screen hierarchy. The query screen activates a shell script and passes on the parameters in the QUERY_STRING. This creates the SQL statement. Once the SQL statement is generated, it returns the data and displays it on the browser via the result screen.

When the QUERY_STRING parameters do not have values (for example when initially activating the shell script), then the query screen defined in the HTML template will be displayed. When the parameters do have values (from either hardcoded or substituted values in the hypertext link that calls the WITS shell script, or from user input on the query screen), then the report will be generated using the header, detail, and footer template types.

8.6.2 Define HTML Templates for Result Screens

Each result screen (presenting query results) is composed of three sections, header, detail, and footer. These are template types within a HTML Template ID. The result screen is where you will enter the HTML and indicate where the data should be placed. A result screen can also accept end user input and act as a query screen.

For details about which substitution tags can be used in each type of screen, see section 4.1.2, Substitution Tags. For details on how to define HTML templates for result screens, see section 10.7.1, Define HTML Templates.

8.7 Define each Data Template

This section gives an overview for design purposes. For details of how to actually perform this task, see Section 10.7.2, Define Data Templates.

Create a group of data template records (uniquely identified by the template id, parameter type, and sequence) to define the basic elements of the query needed to support the data on the report. Each of these records represents a parameter of one of the following types:

- column - a data retrieval field
- where - a query criteria field
- info - informational field passed from screen to screen
- proc - a stored procedure

Either select a parameter from the drop-down list of column aliases, or enter a stored procedure name in the case of the "proc" parameter type.

There are two types of data templates. One is used for dynamic SQL and the other for calling stored procedures. Note that in the data template table those parameter types that are of type INFO function identically for dynamic SQL and stored procedures. Parameter types of COLUMN, and WHERE function differently in dynamic SQL than they do for stored procedures. The following sections identify those differences.

8.7.1 Define Dynamic SQL Data Templates

8.7.1.1 Define Dynamic SQL data columns to be accessed (select clause)

For dynamic SQL, enter the column alias name for the data column in the screen field Parameter. These column alias names will be converted to the physical data column names at the time that the dynamic SQL statement gets generated. Enter the value "COLUMN" in the screen field parameter type, and enter a unique sequence number .

If any of the data to be accessed is defined in the column alias table with an aggregate function then special care must be taken with other columns in the same report. Any columns which are not defined with an aggregate function but appear in the same report must have a "Yes" in the Used In Group By field.

Instead of hard coding a column name, you may wish to allow the end user to choose from a selection list, drop down list box, etc., and have the value they choose included as a column on the dynamic SQL statement. To accomplish this you should enter a "Yes" in the Uses Substitution field of the Data Request Template Screen for that column. When the dynamic SQL is generated, the translation of the column alias will be done based on the parameter value sent in from the HTML screen rather than the parameter listed in the data template table. This is the reason for creating multiple column aliases for the same physical data column.

8.7.1.2 Define Dynamic SQL selection criteria (where clause)

The selection criteria in the WHERE clause of the SQL statement is defined by creating multiple records in the data template table.

For a query with multiple WHERE columns, the end user may choose to query by only some of those where criteria. Any such selection criteria that the end user does not enter a value for will not be included in the where clause of the SQL statement.

The relational operator is required in the WHERE clause for dynamic SQL, but is not used when calling a stored procedure. The WITS Template Builder offers a drop down list box that limits the selection of relational operators available for dynamic SQL.

The Uses Substitution and the Used in Group By fields do not apply to where clauses, they are only applicable to defining the data columns.

8.7.2 Define Stored Procedure Data Templates

A stored procedure should be used for the following reasons:

- to access more than one data table per template, without using a view
- to provide data validation edits, and
- to use WITS to insert, update or delete rows from the database tables.

In the Data Template screen for stored procedures, enter the value PROC in the screen field parameter type, and enter the name of the stored procedure in the screen field parameter value.

8.7.2.1 Define Stored Procedure data columns to be accessed

If you are calling a stored procedure, any COLUMN names defined on the screen will be ignored. Parameters of type COLUMN do not apply to stored procedures because the data columns are defined within the stored procedure. The WITS Template Builder will allow the developer to add COLUMNs to a Data Template for a stored procedure; however, they will not be translated to a physical data column. The resultant data column substitution will instead be controlled by the sequence of columns returned from the stored procedure.

8.7.2.2 Define Stored Procedure selection criteria

WHERE column values entered in the Data Template are passed as parameters to the stored procedure. The relational operator does not apply because the stored procedure is defining how the parameter will be used.

Normally, if no selection criteria are passed in the $QUERY_STRING for the stored procedure, WITS will display the query screen for that template ID. When the stored procedure does not require any parameters, a "dummy" parameter needs to be sent as part of the WHERE criteria in the Data Template so that WITS will generate the report (header, detail, and footer). This dummy parameter must also appear in the QUERY_STRING. Dummy parameters should only be used with stored procedures. If dummy parameters are used with dynamic SQL, then either no data will be returned, or a DBMS error will occur because no such column exists.

9. Design UNIX file structures and Test the application

9.1 Define the migration environment.

WITS is a tool for you, the software developer, to use in developing applications. The definitions specified in the WITS template tables (the definition of your application) should be thoroughly tested before being released to a production environment. The functions that drive WITS will not abort, but may instead result in an error message. In any case, it is necessary to validate definitions.

Often it is desirable to have not only the development and production environments but also a staging area between them. The WITS system provides a set of parameters that accommodate the need to migrate applications from one environment to another.

You may want to define a set of parameters for each stage of your migration environment. The environment is a set of paths to your shell scripts, HTML documents, WITS Metadata tables, Host Server Locations, Shell Script parameters, and Data Group table parameters.

Figure 9.1—1 WITS Production Environment Values for FIS, describes the current production environment of the WITS application for the FIS data group. This same format can be used to document the values for the development and quality (testbed) environments.

Figure 9.1—2 WITS Environment Variables Form can be used to document the values for the other migration environments.

Data Group: FIS **Environment: Production**

Description	Parameter	Environment Values
ACCESS FILE PARAMETERS:		
Shell Scripts:	URLPATH1=	http://www-irn.sandia.gov/cgi-bin/templates/
filename1		template_fis.sh
HTML Documents	URLPATH2=	http://www-irn.sandia.gov/template_html/
htdocname1		fishelp.html
htdocname2...		
WITS Template Tables	SERVER=	prod = sybgila
	UID=	mosaic
	PWD=	mosaic1
	DBNAME=	qryparmsdb
Application Values	APPNAME=	TotalCost
Seconds for attempted login	LOGINTIMEOUT=	3
Seconds for attempted query	SQLTIMEOUT=	60
Message embedded in HTML	BCASTMSG=	<blink>New Web Application</blink>
Formatting of money fields	PRINT$=	1
Host Server Locations:		
Sybase Interfaces file	INTERFACES=	/usr/sybase/interfaces
Log Files	LOGPATH=	/usr/netsite-cgi/templates/logs/
Sockets	SOCKETPATH=	/users/netsite/
QRP Function (path and executable name)	DATASERVERPROG=	/usr/netsite-cgi/templates/bin/qrp
SHELL SCRIPT PARAMETERS:		
WTP Function (path and executable name)	(launch command)	/usr/netsite-cgi/templates/bin/wtp
Logging option indicator	-l	
AccessFile (path and filename)	-a	/usr/netsite-cgi/templates/access/AccessFis
Default Template Name	-t	CaseOrgTotals
Query String variable	-q	$QUERY_STRING
DATA GROUP TABLE PARAMETERS:		
User Application Data tables	DBSERVER	prod = sybgila
(parameters correspond to the	USERID	mosaic
values for access_parm_id)	PASSWORD	mosaic1
	DBNAME	dwdb
	INTERFACES	/usr/sybase/interfaces

Figure 9.1—1 WITS Production Environment Values for FIS

Data Group: _____ **Environment: _____**

Description	Parameter	Environment Values
ACCESS FILE PARAMETERS:		
Shell Scripts:	URLPATH1=	
filename1		
HTML Documents	URLPATH2=	
htdocname1		
htdocname2...		
WITS Template Tables	SERVER=	
	UID=	
	PWD=	
	DBNAME=	
Application Values	APPNAME=	
Seconds for attempted login	LOGINTIMEOUT=	
Seconds for attempted query	SQLTIMEOUT=	
Message embedded in HTML	BCASTMSG=	
Formatting of money fields	PRINT$=	
Host Server Locations:		
Sybase Interfaces file	INTERFACES=	
Log Files	LOGPATH=	
Sockets	SOCKETPATH=	
QRP Function (path and executable name)	DATASERVERPROG=	
SHELL SCRIPT PARAMETERS:		
WTP Function (path and executable name)	(launch command)	
Logging option indicator	-l	
AccessFile (path and filename)	-a	
Default Template Name	-t	
Query String variable	-q	
DATA GROUP TABLE PARAMETERS:		
User Application Data tables	DBSERVER	
(parameters correspond to the	USERID	
values for access_parm_id)	PASSWORD	
	DBNAME	
	INTERFACES	

Figure 9.1—2 WITS Environment Variables Form

9.2 Create a Shell Script.

The shell script is the top level function of an application, used to launch the WITS runtime system. Prepare a shell script in the location specified for the URLPATH1 for the desired environment.

For details about how to create the shell script see section 5.1, UNIX Shell Script with $QUERY_STRING and Command Line Parameters.

9.3 Create an Access File

The access file contains parameters used by the WITS run-time system to connect to the DBMS, locations of WITS components, print formats, banner messages, URL paths, and time out specifications. Prepare an access file in the location specified in the shell script.

For a detailed explanation of all Access File parameters, see section 5.4, Access File with Parameters.

9.4 Test Your Application

The WITS Template Builder creates and modifies application definitions that are stored on the development Sybase server. All queries performed during testing are performed against user data on the same database server. The functional environment could also be located on the development host by adding your shell script and access file to that machine.

To test your application, open a Web browser and initiate your Web application with a URL to the UNIX shell script you have created. If the URL is not embedded as hypertext in a document, then set a bookmark for the URL. In order to revise the look of the HTML page, open the WITS Template Builder and make changes to your application definition, then switch back to the browser and click on the Reload icon.

When your application is fully tested, the DBAs should be notified to move the data views for your data group from the Sybase server in your current environment to the next Sybase migration environment. When that is done you will need to change your access file to point to the production Sybase server. The functions already exist on the production Web server, but your access file and shell script will have to be copied to the production server. The production URL to launch your application should point to the new shell script to be executed, which will point to the access file, which will now point to the production Sybase server for both the WITS data and user data.

10. How to Perform Tasks in the WITS Template Builder

10.1 How to start up the WITS GUI front end (WITS Template Builder)

On your desktop, double click the WITS icon (which was set up on your desktop during the WITS installation procedure).

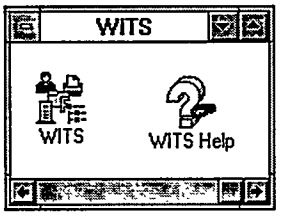


Figure 10.1—1
WITS Icon

10.2 How to Login to the Development Region

The Database Login dialog box pops up when you first open the WITS Template Builder.

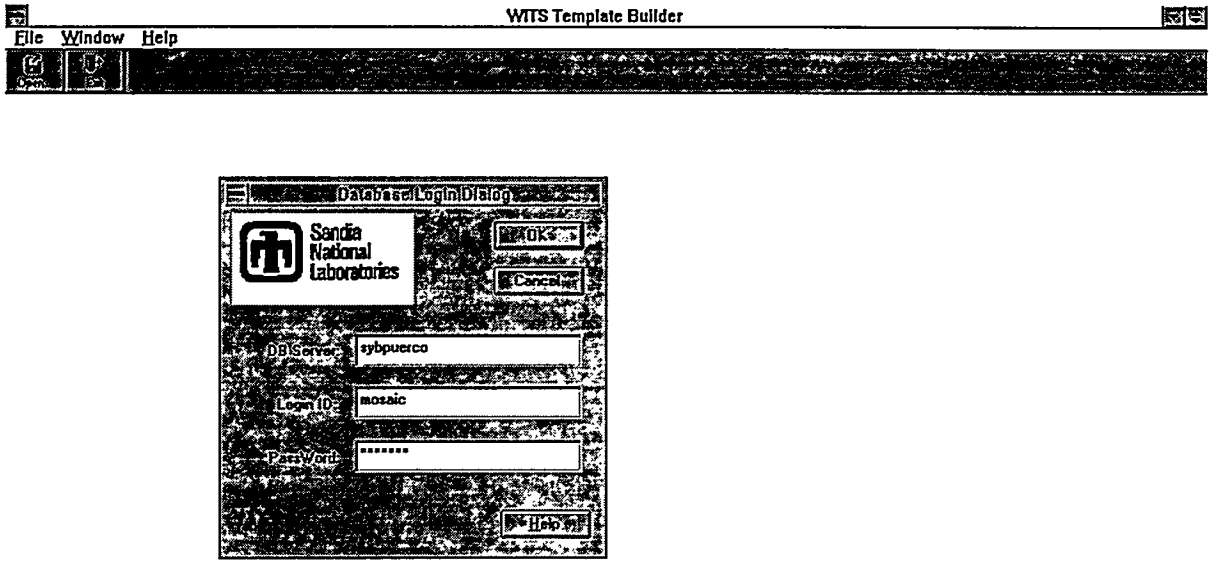


Figure 10.2—1 WITS Login Dialog Box

Fill in the database server, login ID, and password fields of the Database Login dialog box. The database server name is not case sensitive, but the login id and password are case sensitive. The password is mandatory. The DBServer and Login Id are optional if these values exist in your pb.ini file. Click the OK button or press Enter. If your login is correct, the Data Groups screen will be displayed.

If your password is blank or incorrect, or if the DBSERVER or Login ID was not found, or are incorrect, WITS will be unable to connect to the database, and will not display the Data Groups Screen. WITS will display an error message dialog box. If you click the cancel button WITS will terminate.

10.3 The Data Group Window

The Data Group window is automatically displayed when you successfully log in. Use it as the starting point for all WITS Template Builder tasks. The left side of the window contains a list of all the WITS Data Groups that have been defined. The right side has two groupings, **Data Group** Insert or Edit, and **Definitions** for Data Access and Templates.

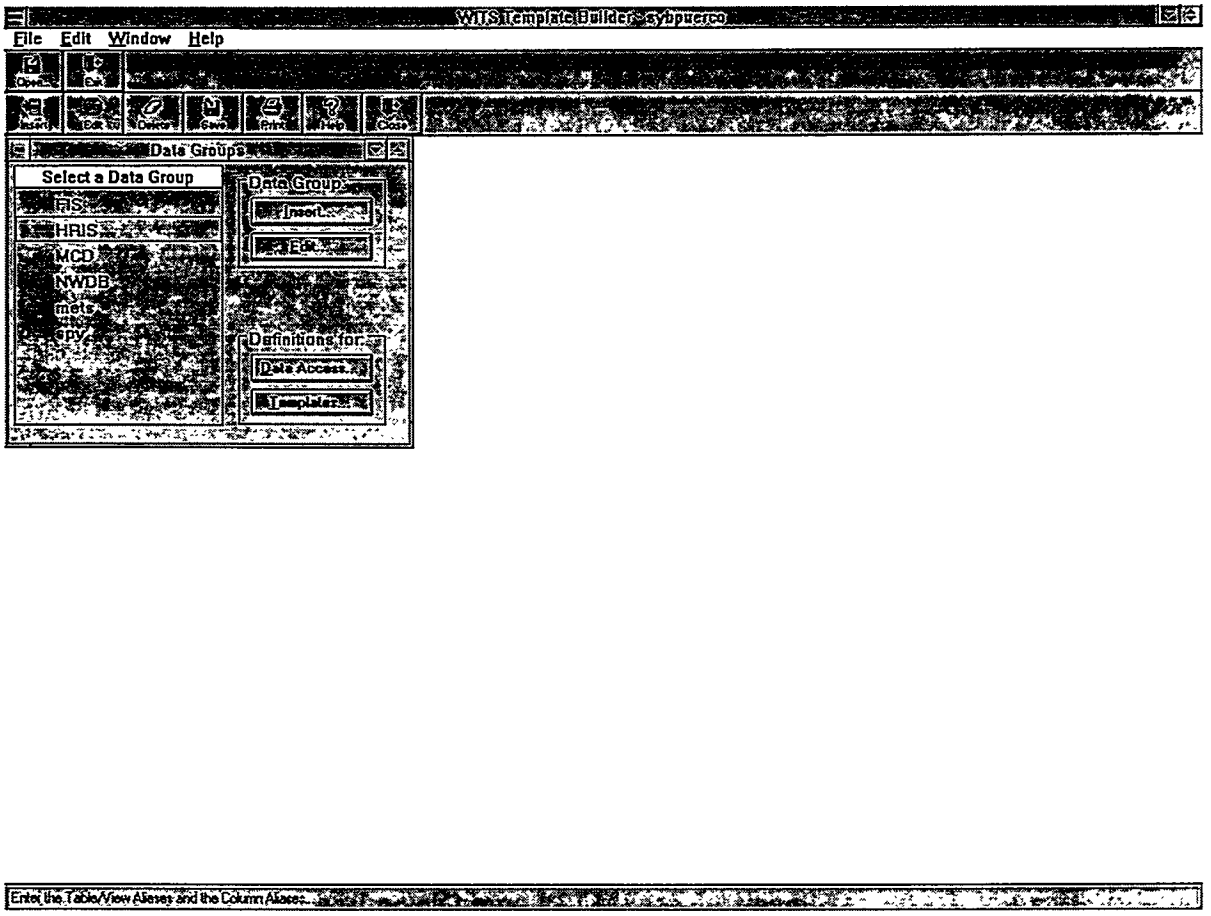


Figure 10.3—1 Data Group

You have four options to select from:

1. Insert - Insert a new Data Group into the list. Remember, a Database Administrator will need to create the views for any new data groups and give you update access to them. For details, see section 10.4, Inserting a new Data Group.
2. Edit - Change the Access Parameters for the selected data group. See section 10.5, Access Parameters.
3. Data Access Definitions - define the table and view aliases and the column aliases needed to access your data. See section 10.6
4. Template Definitions - define the HTML templates and Data templates needed to retrieve and present your data. See section 10.7

10.4 Create, Change, and Delete Data Groups

When you insert or edit a Data Group, the Data Group table is being manipulated . For details on the Data Group table, see section 6.1.3, data_group Table.

Insert a new Data Group:

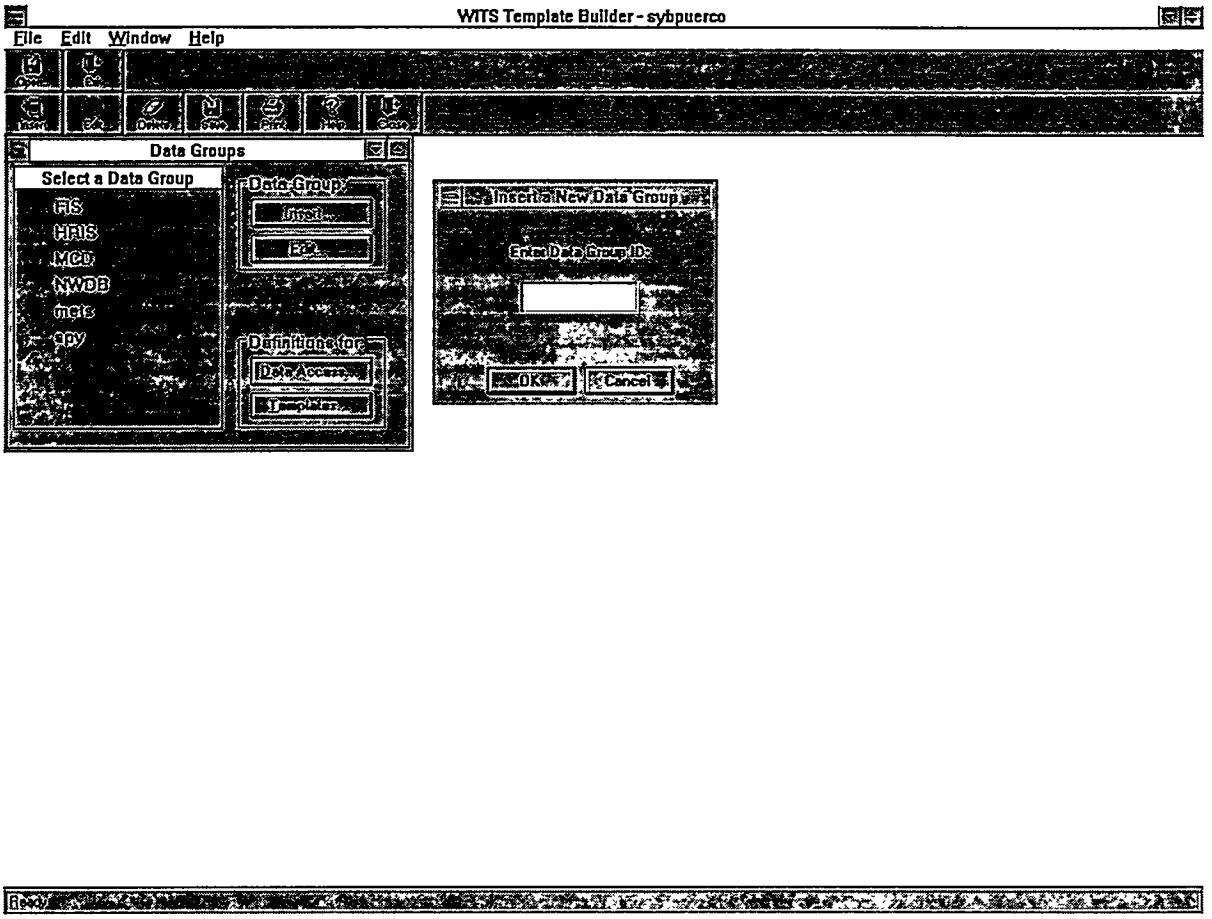


Figure 10.4—1 Insert a New Data Group

To create a new Data Group:

- Click the Insert button on the Data Groups window
- Type the new name in the Insert a New Data Group window. The name is case sensitive.
- Click the OK button
- In the Access Parameters window (see Figure 10.4—2 Change a Data Group), enter a value for each Access Parameter. They are case sensitive.
- Click the OK button.

Change a Data Group:

Change the Access Parameters (DBNAME, DBserver, interfaces, password, user ID) for a Data Group with the Access Parameters window. This is necessary when you migrate from one environment to another, that is, from development to quality (testbed), or from quality (testbed) to production.

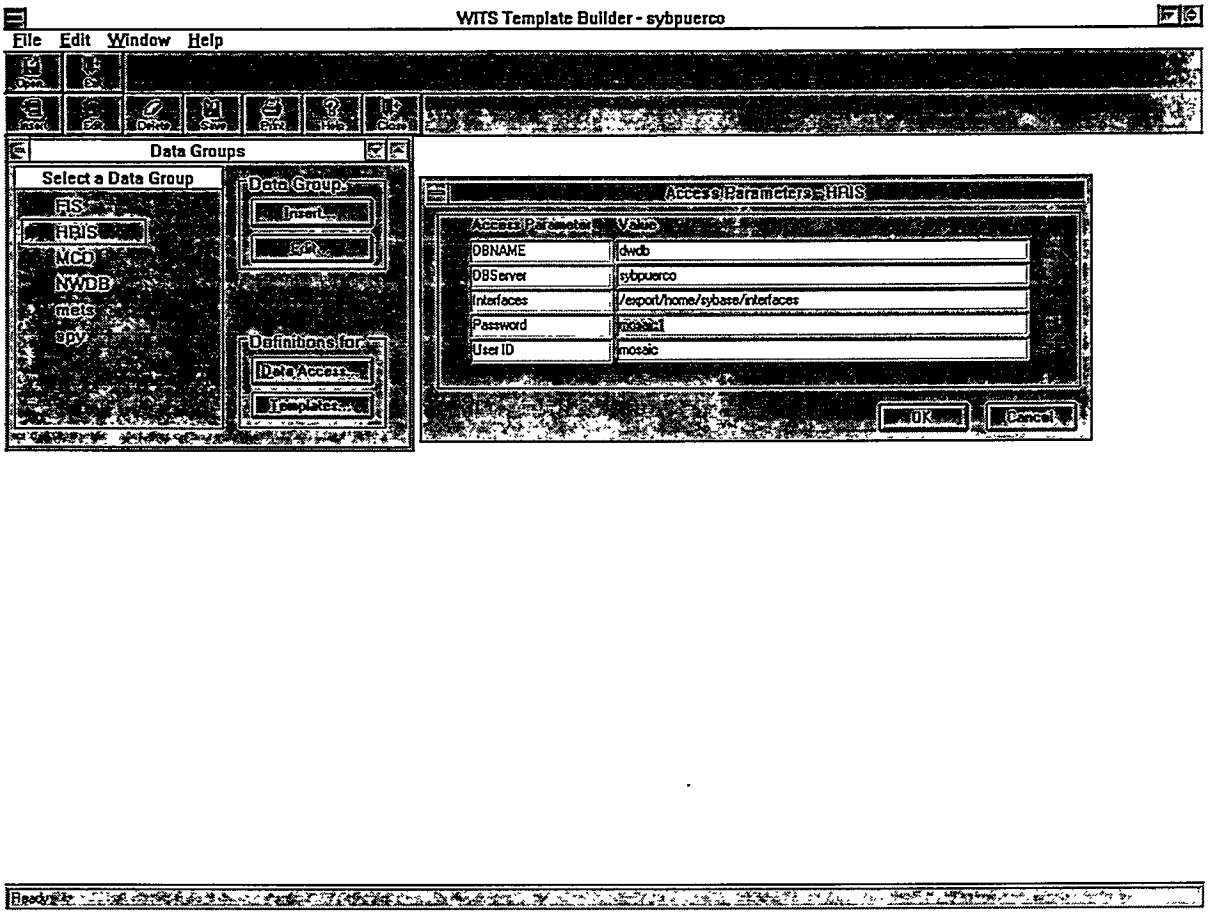


Figure 10.4—2 Change a Data Group

To change a data group's access parameters:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Edit button on the Data Groups window
- In the Access Parameters window, change the values for the Access Parameters. They are case sensitive.
- Click the OK button.

Deleting a Data Group

To delete a Data Group you must contact a DataBase Administrator.

10.5 Establish Data Access for a Data Group

Establishing data access for a data group involves defining aliases for views and aliases for columns (or functions of columns). These are called Table/View Aliases and Column Aliases.

10.5.1 Establish a Table/View Alias

The WITS QRP uses the table/view alias during the creation of its dynamic SQL. It takes a table/view alias and finds the real database name and table name. Dynamic SQL requires at least one alias for each table/view in a database. Additional aliases could be used to access the same physical table under multiple conditions.

When you establish table/view aliases for a Data Group, the View Location table is being manipulated. For details on the View Location table, see section 6.1.7, view_location Table.

If you are using a stored procedure as the sole access to the data, then you do not need to establish an alias for the table/view. Skip this section. You would use a stored procedure as the sole access if your application will access more than one data table in a single template, and a view would not support the needed functionality.

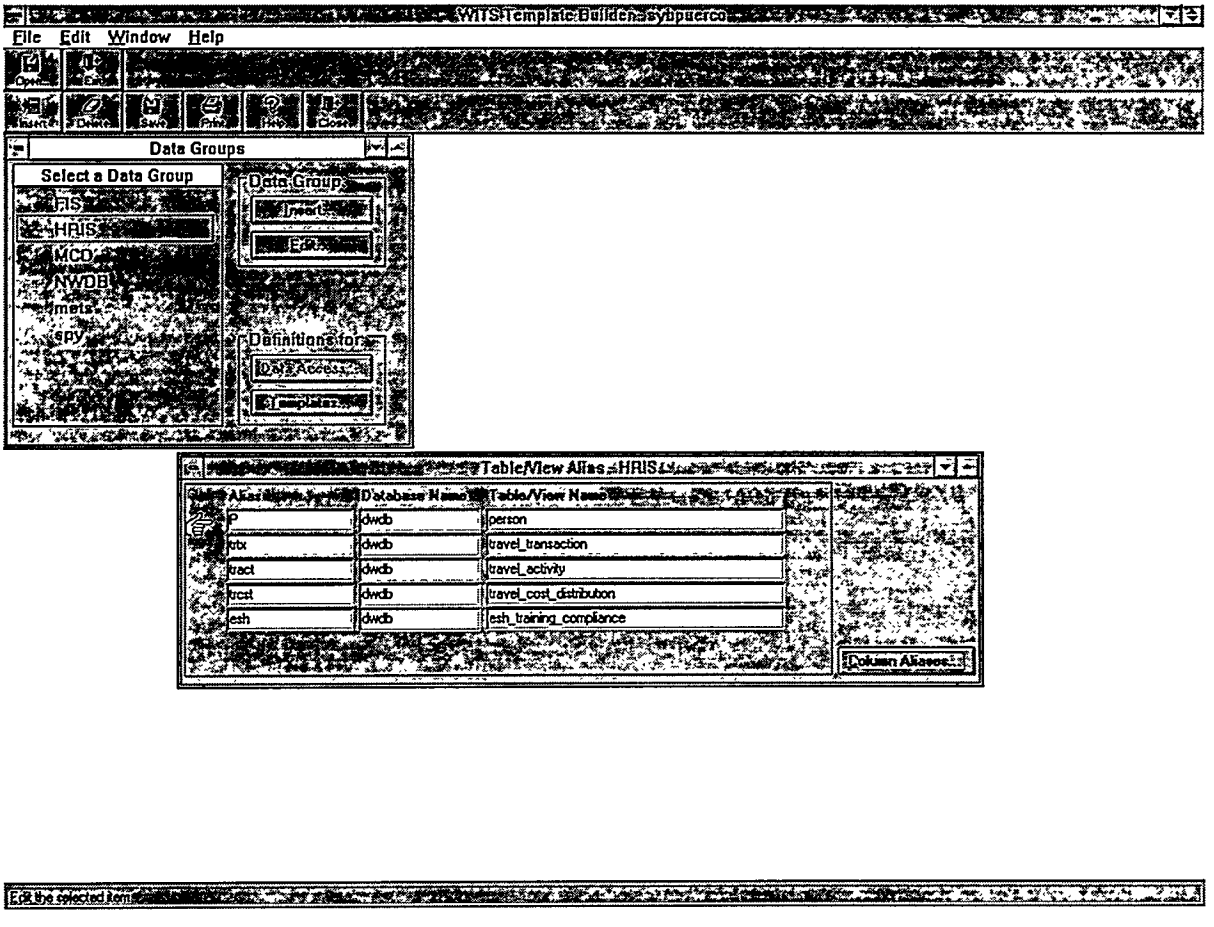


Figure 10.5—1 Table Alias

To insert a new Table/View alias:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Data Access button in the Data Groups window.
- Click the Insert icon in the secondary toolbar.
- In the empty row that is inserted in the Table/View Alias window, type the new alias, database name, and table/view name.
- Click the OK button.

To change a Table/View alias:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Data Access button in the Data Groups window.
- In the Table/View Alias window, change the alias for a database name and table/view name.
- Click the OK button.

To delete a Table/View Alias:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Data Access button in the Data Groups window.
- In the Table/View Alias window, select the desired alias.
- On the secondary toolbar, click the Delete icon.
- Click the Yes button in the message dialog box

10.5.2 .Establish a column alias

The column alias allows for a translation from application parameters to physical database column/function names. It allows database functions to be called transparently, by using an alias name. It also allows the application to dynamically choose, at run time, the column to be retrieved, based on user input. A column alias (or column/function alias) is needed for each unique function of a DBMS column.

The Column Function Alias table is being manipulated when you create, change, or delete a column alias. For details about this table , see section 6.1.6, column_function_alias Table.

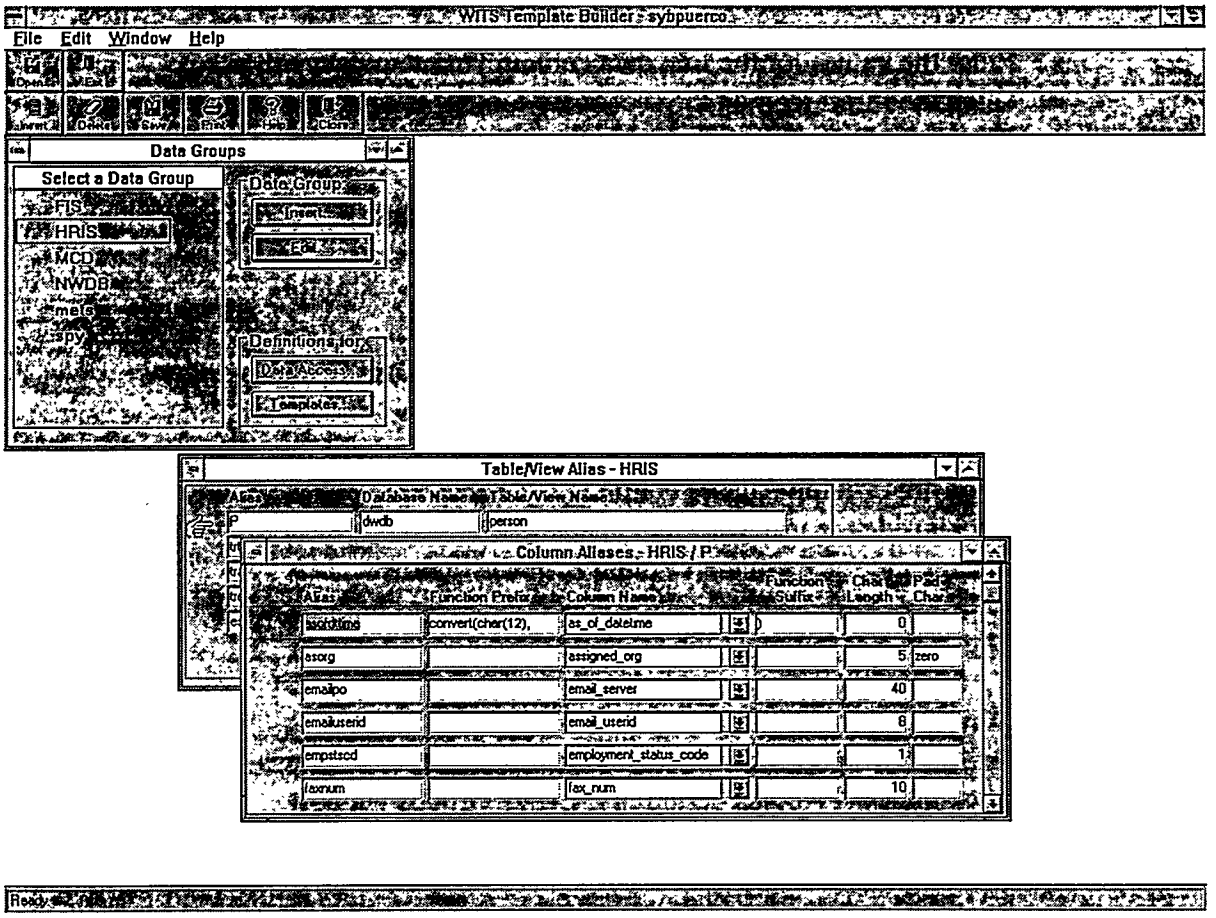


Figure 10.5—2 Column Alias

To insert Column Aliases:
One or more column aliases will already exist for previously defined Template IDs. To insert additional aliases:

- Select (highlight) the desired data group in the Data Groups window,
- Click the Data Access button in the Data Groups window
- Select the desired alias in the Table/View Alias window.
- Click the Column Aliases button in the Table/View Alias window.
- Click the Insert icon in the secondary toolbar.
- Fill in the values in the Column Aliases window.

If you have just created a new Table Alias, and click the Column Aliases button on the Table/View Alias window, the Column Aliases window will open in insert mode, and a blank row will be displayed.

- Enter the values for the first data column your template will access.
- Click the Insert icon on the secondary toolbar to open another blank row.
- When you have finished entering data columns, close the Data Template window.
- Click the Yes button to save the Column Alias. (Note: If you did not insert any Column Alias rows, click the No button rather than trying to save a blank Column Alias).

To change Column Aliases:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Data Access button in the Data Groups window
- Select the desired alias in the Table/View Alias window.
- Click the Column Aliases button in the Table/View Alias window.
- Click inside the row you want to change.
- Change the values in the Column Aliases window.
- Click the Close button on the secondary toolbar or double-click in the control box.
- Click the Yes button in the message dialog box to save.

To delete Column Aliases:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Data Access button in the Data Groups window.
- Select the desired alias in the Table/View Alias window.
- Click the Column/Aliases button in the Table/View Alias window.
- Click inside the row you want to delete.
- Click the Delete icon on the secondary toolbar.
- Click the Close button on the secondary toolbar or double-click in the control box.
- Click the Yes button in the message dialog box to delete.

10.6 Define Template IDs

Template IDs allow the WITS QRP to find out what database to access. They also set the maximum number of rows that can be retrieved from a query.

When you create, change, or delete Template Ids you are manipulating the Data Request Table. For details about this table see section 6.1.4, data_request Table.

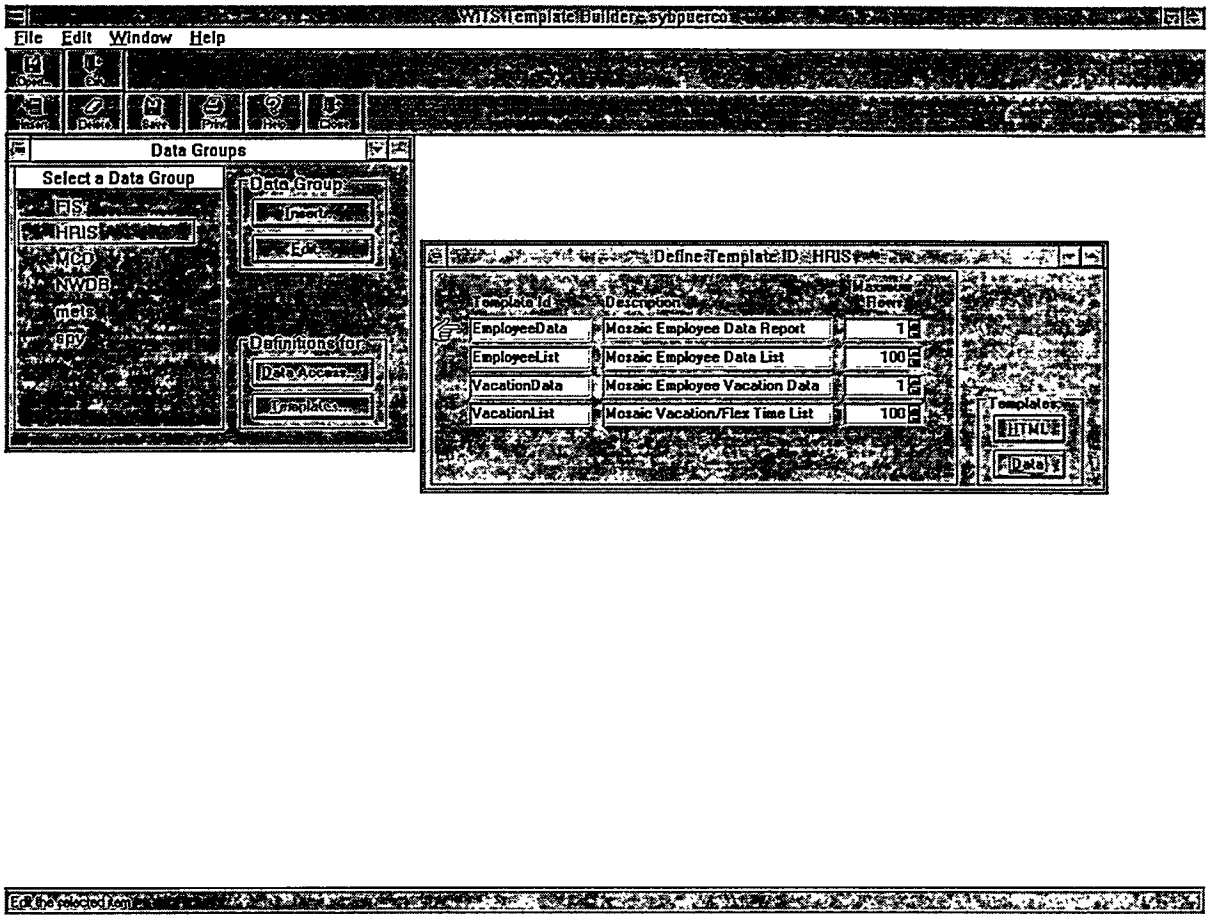


Figure 10.6—1 Template Ids

To insert a new Template ID:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Template button in the Data Groups window.
- Click the Insert icon in the secondary toolbar.
- In the empty row that is inserted in the Define Template ID window, type the new Template Id, Description, and maximum rows.
- Close the Template ID window.
- Click the Yes button on the message dialog box to save.

To change a Template ID:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Template button in the Data Groups window.
- Type new values for the Description or maximum rows in the Define Template ID window.
- Close the Template ID window.
- Click the Yes button on the message dialog box to save.

To delete a Template ID:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Template button in the Data Groups window.
- Select the desired alias in the Define Template ID window.
- On the secondary toolbar, click the Delete icon.
- Click the Yes button in the message dialog box to delete.

10.7 Define Templates

Each Template ID owns two types of templates, HTML Templates and Data Templates.

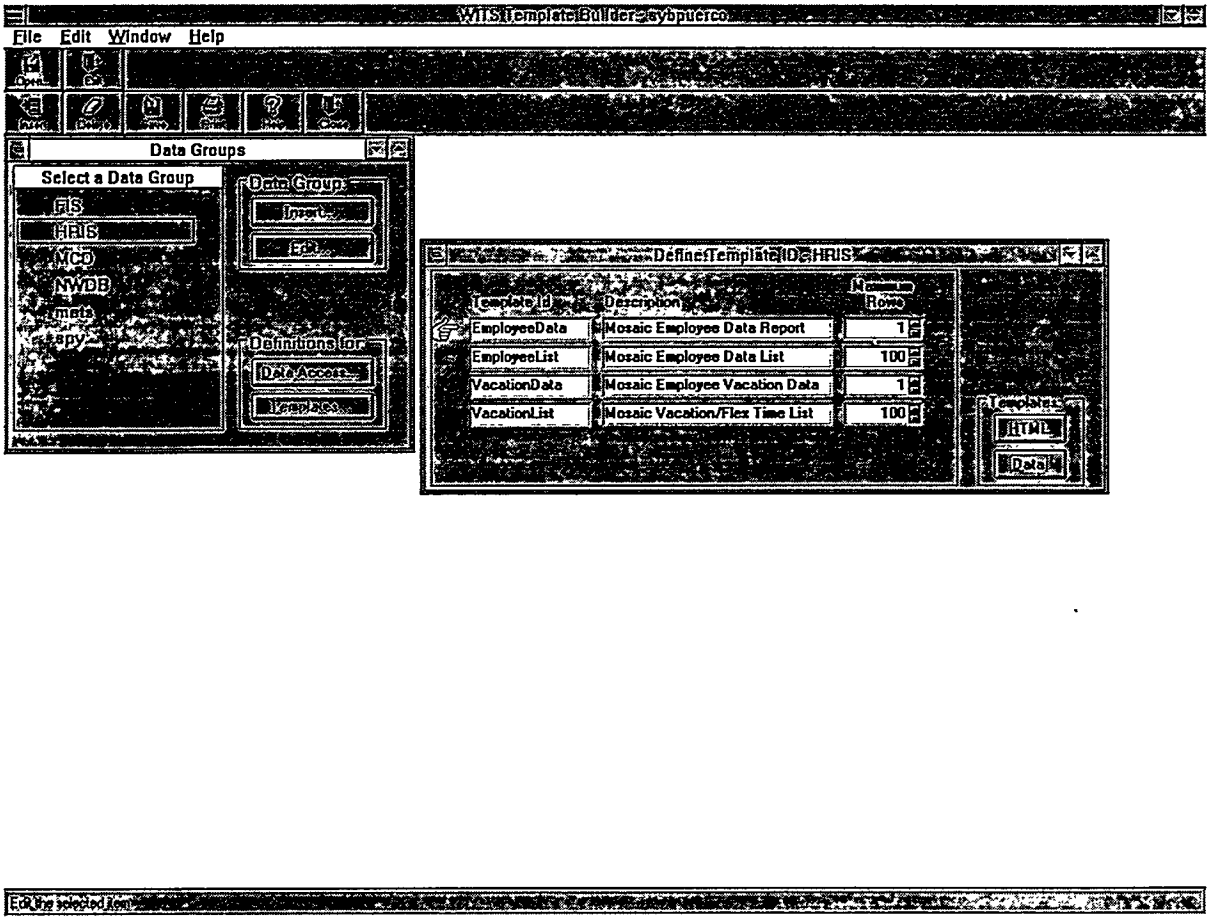


Figure 10.7—1 Choosing HTML or Data Templates

To choose to work on HTML or Data Templates:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Templates button in the Data Groups window
- Select the desired Template ID in the Define Template ID window.
- Click the HTML button in the Define Template ID window.

10.7.1 Define HTML Templates

HTML templates contain the HTML markup language and substitution tags for the query screen and for the header, detail, and footer portions of the results screen. When you insert, change, or delete a HTML template, you are manipulating the Template HTML table. For details about that table, see section 6.1.6, template_html Table.

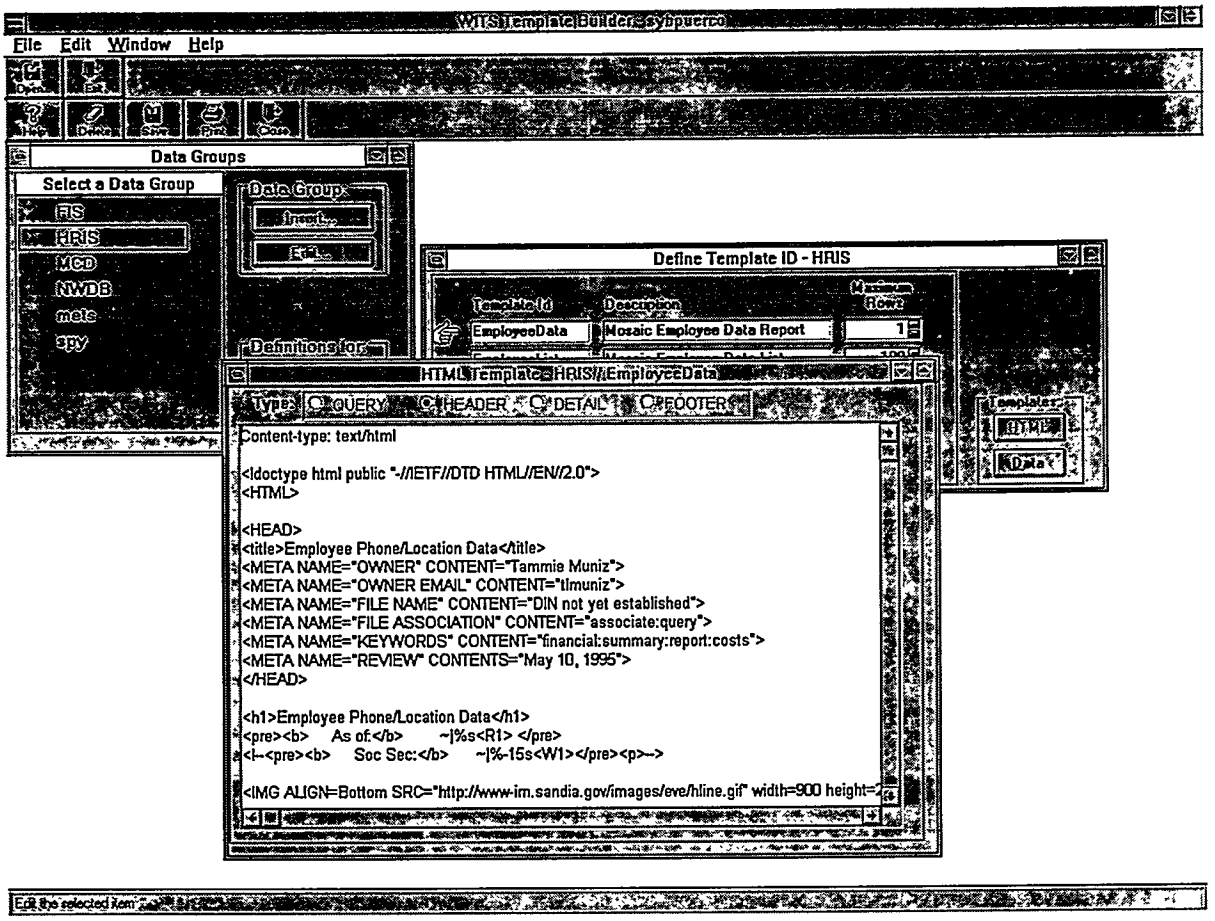


Figure 10.7—2 HTML Template

To insert an HTML Template:

WITS inserts a blank HTML Header type template when you create a new Template ID.

To change an HTML Template:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Templates button in the Data Groups window
- Select the desired Template ID in the Define Template ID window.
- Click the HTML button in the Define Template ID window.
- Verify, in the HTML Template Window, that the desired HTML template type is selected (header, footer, detail, query).
- Change the HTML template as needed. Copy and paste options are available under the WITS Edit menu.

To delete an HTML Template:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Templates button in the Data Groups window
- Select the desired Template ID in the Define Template ID window.
- Click the HTML button in the Define Template ID window.
- Verify, in the HTML Template Window, that the desired HTML template is selected (header, footer, detail, query).
- Click the Delete icon on the secondary toolbar.
- Click the Yes option on the message dialog box to delete.

10.7.2 Define Data Templates

Data templates allow the QRP to convert the query string (which came from the HTML page through the Shell Script or was hardcoded in the URL) into a command string for the Data Server. Data templates can also be used by non-Web applications to identify variables.

Inserting, changing, or deleting data access templates manipulates the Template Data Request table. For details about the Template Data Request table see section 6.1.5, template_data_request Table.

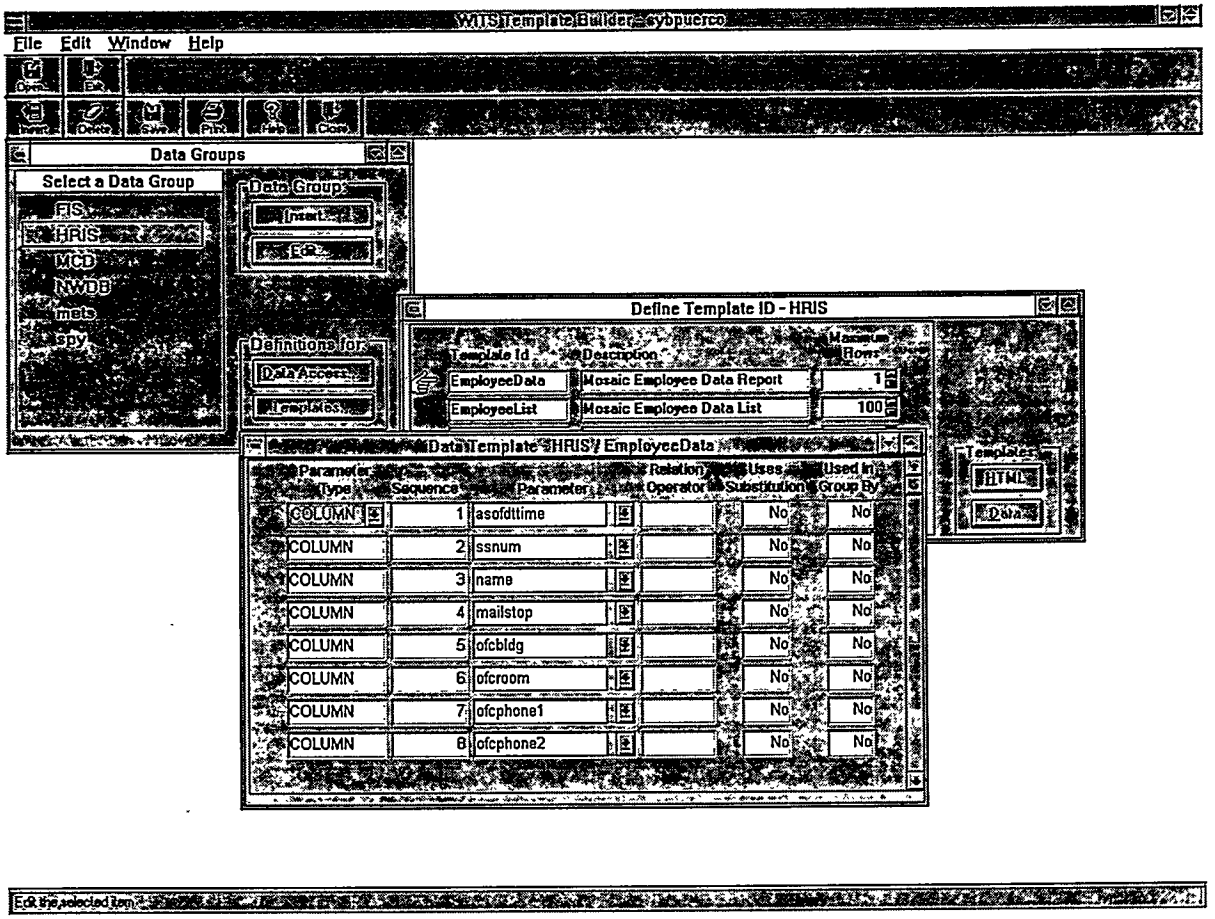


Figure 10.7—3 Data Templates

To insert rows in a Data template:

A data template will already exist for previously defined Template IDs. If you have just created a new Template ID, and click the Data button on the Define Template ID window, the Data Template window will open in insert mode, and a blank row will be displayed.

- Enter the values for the first data column your template will access.
- Click on the Insert icon on the secondary toolbar to insert additional blank rows.

- When you have finished entering data columns, close the Data Template window.
- Click the Yes button to save the Data Template. (Note: If you did not insert any rows on the new Data Template, click the No button rather than trying to save a blank Data Template).

To edit rows in a Data template:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Templates button in the Data Groups window
- Select the desired Template ID in the Define Template ID window.
- Click the Data button in the Define Template ID window.
- Select the desired row in the Data Template window.
- Select the desired parameter type (where, column, info, proc).
- Change the values as needed.

To delete rows in a Data template:

- Select (highlight) the desired data group in the Data Groups window.
- Click the Templates button in the Data Groups window
- Select the desired Template ID in the Define Template ID window.
- Click the Data button in the Define Template ID window.
- Select the desired row.
- Click the Delete icon on the secondary toolbar.
- Click the Yes button in the message dialog box to delete.

11. WITS Projects - Completed, In Development, and Future

11.1 Completed projects built using WITS

11.1.1 Financial Management Reporting for Web Browsers

WITS was developed as part of the Financial Management Reporting system, but with an emphasis on creating a reusable tool for future software development projects. The system uses a Web interface to accept requests for cost budget data such as Case, Charging Organization, Management Code, and Funding Code. Data is reported on either a monthly basis or a fiscal year to date basis. It can be summarized at any level of Case or Charging organization.

11.2 Web Projects Currently in Development using WITS

WITS is being used to define an HTML interface for the following projects:

11.2.1 Network Database (NWDB)

The Network Database is used by Network Administrators and Customer Service Units. This database is Sandia's system for keeping track of UNIX User IDs, login names, networks, LANs, LAN connections, machines, machine security plans, central service accounts, security plans for major networks and LANs, E-mail post offices, Internet domain name services, and related information.

The input portion of this system is launched from the Web but executes as an Extensible Virtual Toolkit (XVT) client/server application. The output portion uses WITS to create output reports on the client's Web browser, with Print and Save As capabilities provided by the browser.

11.2.2 Systems Development Lifecycle Metadata Repository

The repository assists in the administration of software design life cycle components. The repository provides a catalog of data and function components to improve understanding and reusability. It also performs some of the needed transformations among the various phases and components of the life cycle. It stores this information in a set of relational tables and is accessible through a transaction interface as well as WITS.

11.2.3 Mail Channel (MCD)

The Mail Channel Directory provides addresses for sending classified mail and materials to locations outside SNL. The address data is maintained in Sybase using an Access application. The Mail Channel Directory Search can be accessed on Sandia's Internal Web.

It uses the WITS system to provide a basic query listing and hyperlinks to detail on each Mail Channel.

11.2.4 Manufacturing Information Service Requests (MISR)

The Manufacturing Information Service Request tracks customer requests for programming services. This application will use WITS and a stored procedure to allow customers to input their requests through the Internal Web. It will use a WITS query screen to allow for searching.

11.3 Non-Web Projects Currently in Development using WITS

11.3.1 Mail Enabled Access to FIS Data

The Mail Enabled Access project allows users to send a request for financial data via e-mail, and receive that data back via e-mail (any e-mail system that supports attachments, such as cc-mail, Microsoft Mail, WordPerfect Office, etc.). Another version of it uses an HTML interface on the Web to send the e-mail request. The function which processes requests and sends data back to e-mail is being redeveloped in C to use the WITS Query Retrieval Processor without the WITS Web Interface Processor. This would allow users to send an e-mail request for data and receive that data from the Data Warehouse, eliminating all the e-mail data that is currently in FoxPro tables.

11.4 Future Projects that Might use WITS

Any system that will be retrieving data from a Sybase database and presenting that data via Web browser software could use WITS now as a development tool.

The WITS QRP could also be used without the WTP to retrieve Sybase data and present it in some format other than HTML. In the future WITS could also be used to retrieve data from other relational databases in addition to Sybase. See section 12, WITS Future Enhancements.

Some projects that are potential candidates to use WITS should include the following:

11.4.1 PDI (Project Data Interface)

This Visual Basic based project tracks project plans. It may be converted to HTML to make it accessible on the internal web, and will use the Query Retrieval Processor (accessing the financial data) to eliminate PDI's current large data downloads from the mainframe.

11.4.2 GraFin

This project takes current financial case data from the Laboratory Information System machine (LIS), analyzes the data, and produces graphs in Microsoft Excel. It could be converted to get data from the data warehouse via WITS.

11.4.3 Spend Plan

Spend plan is application that allows users to enter Spend Plan Adjustments (initial input included) on their PC or Macintosh computer. The Tool creates files that can then be e-mailed to the appropriate Primary Management Area (PMA) business office. The PMA office reviews adjustments and upon approval uploads the file(s) to the Financial Information System (FIS) for processing and posting. The Spend Plan Tool was a joint development effort between Sandia National Lab (SNL) New Mexico and SNL California.

The tool was developed in FoxPro 2.6 and is able to run on both PC and Mac platforms. Spend Plan could be modified to use WITS to access financial data instead of e-mailing this data to each user.

11.4.4 Operational Planning

Operational Planning is a new PowerBuilder™ application that collects data about future operations, and analyzes it for the combined "Operational Plan". It uses financial data which could be obtained with an interface to WITS Query Retrieval Processor.

12. WITS Future Enhancements

- Add the ability to send data to the user in other ways than a Web browser interface. For example, developers could use WITS for data access, then send the data to the requester via e-mail, ftp or other methods.
- Add the ability to retrieve data from other relational databases in addition to Sybase.
- Add the ability to dynamically span different databases with a single query.
- Add the flexibility to pre validate input parameter values.
- Expand database connectivity to support multiple concurrent sessions.
- Create a new process to replace Mail Enabled that will interface to the QRP.
- Add support for additional relational operators that are selectable by the end user.
- Add a database logging feature for compiling application metrics and incident tracability.
- Add the capability to dynamically join tables according to WITS definitions. This would increase the power of WITS, allowing transparent access to columns from multiple tables.

DISTRIBUTION:

	MS 9018	Central Technical Files, 8523-2
5	MS 0899	Technical Library, 13414
	MS 0619	Print Media, 12615
2	MS 0199	Document Processing, 7613-2 For DOE/OSTI
	MS 0630	Michael J. Eaton, 4000
	MS 0803	John F. Jones, 4600
	MS 0629	Michael G. Robles, 4800
	MS 0622	L. Herbert Pitts, 4400
	MS 0801	Melissa J. Murphy, 4900
	MS 0898	R. Dennis Rowley, 4812
	MS 1098	Thomas L. Ferguson, 4813
	MS 1090	Joseph A. Ruggles, 4814
	MS 1090	Grant C. Claycomb, 4815
	MS 0661	Gary Rivord, 4816
	MS 0661	Michael H. Pendley, 4612
	MS 0813	Jim Hamilton, 4412
	MS 0807	R. Michael Cahoon, 4918
	MS 0622	Hank Witek, 4606
	MS 1090	William D. Swartz, 4411
	MS 0812	Sharon Trauth, 4923
	MS 0806	Michael O. Vahle, 4616
	MS 0622	Gerald Esch, 4401
	MS 1098	Jennie Negin, 4403
	MS 0809	George Connor, 4421
10	MS 1098	Lois Lauer, 4813
	MS 1098	Mark Lynam, 4813
	MS 1098	Tammie Muniz, 4813
	MS 1098	Andrea Cassidy, 4403
	MS 1098	Mary Roehrig, 4813
	MS 0812	Ken Osburn, 4922
	MS 0812	Linda Garcia, 4922
	MS 0661	John Hatley, 4816
	MS 0661	Fran Current, 4612
	MS 1012	Alan Armentrout, 3050
	MS 0661	Elisa Kephart, 4816
	MS 1098	Pat Milligan, 4813
	MS 0661	Dave Cuyler, 4922
	MS 1090	Clayton Pryor, 4814
	MS 1098	Scott Rogers, 4813
	MS 0661	Greg Conrad, 4816
	MS 0661	Scott Joyce, 4816
	MS 0813	Mike Finley, 4412
	MS 0813	Bev Ortiz, 4403
	MS 0805	Nancy Marsh, 4911
	MS 0113	Carol Christensen, 10506
	MS 0944	Karen Rogers, 7901