Erik E. Westbrook, Waiman Cheung, and Gilbert Babin. "An Object-Based User Interface for Manufacturing Information Integration." in Rensselaer's Third International Conference on Computer Integrated Manufacturing. Troy, New York, USA. May 1992. pp. 114-118.

# An Object-Based User Interface for Manufacturing Information Integration

**Eric E. Westbrook**
Digital Equipment Corporation
Maynard, Massachusetts;
Rensselaer Polytechnic Institute
Troy, New York

**Waiman Cheung**
Lecturer
Chinese University
of Hong Kong
Hong Kong

**Gilbert Babin**
Department of Decision Sciences
and Engineering Systems
Rensselaer Polytechnic Institute
Troy, New York

**Abstract**

*User interface tool libraries such as Motif [tm] from the Open Software Foundation (OSF) are becoming increasingly prevalent as standards of form and function for highly functional and attractive graphic user interfaces. However, the software knowledge required to create and manipulate reasonable user interfaces using these tools is very significant in size and scope, and can become pervasive and limiting to application development. In this paper, we present a new tool set (WPTSER) that dramatically reduces the amount of knowledge needed to produce a functional and attractive interface which can be executed interchangeably on OSF/Motif [tm] or character-cell terminals. WPTSER also provides user input validation through automatically-generated deterministic finite automata from developer-provided regular expressions. To show the capabilities of WPTSER, we use a global query system developed for a Computer Integrated Manufacturing (CIM) system at Rensselaer.*

## 1: The problem

The design of user interfaces and input validation often accounts for a regrettably large proportion of application development time. In today's computing environment, application development must overcome the following three fundamental challenges:

1. Heterogeneity among terminal hardware and software;
2. Prohibitive complexity of user interface software tools; and
3. The classic need for full validation of user input.

As the importance of rapid and efficient software development increases, the importance of reducing the impact of each of these three issues on application development increases directly.

When combined, these issues form an even more significant barrier to efficient software engineering.

## 2: The *WPTSER* solution

The *Windowing Package for the Two-Stage Entity-Relationship* [1] was created to address these problems. *WPTSER*'s originally focused on the minimization of development effort by providing a facility for the automatic control of user input syntax, thus enabling the developer to concentrate on computational details of the application. Today, *WPTSER* offers a fundamental set of User Interaction Objects (UI Objects) that are functionally consistent when used on all supported types of conventional user terminal equipment. These UI Objects can then be used as building blocks by the application to create the user interface. Through the reduction of formal parameters and the standardization of object behavior, each *WPTSER* UI Object reduces the software effort required to construct and operate an intelligent user interface. While a small degree of low-level graphics customizability is naturally sacrificed in achieving this ambitious goal, significant customizability is nevertheless retained (for those few who require it) through OSF/Motif[tm] window manager resource options which are supported by *WPTSER*.

## 3: *WPTSER* architecture

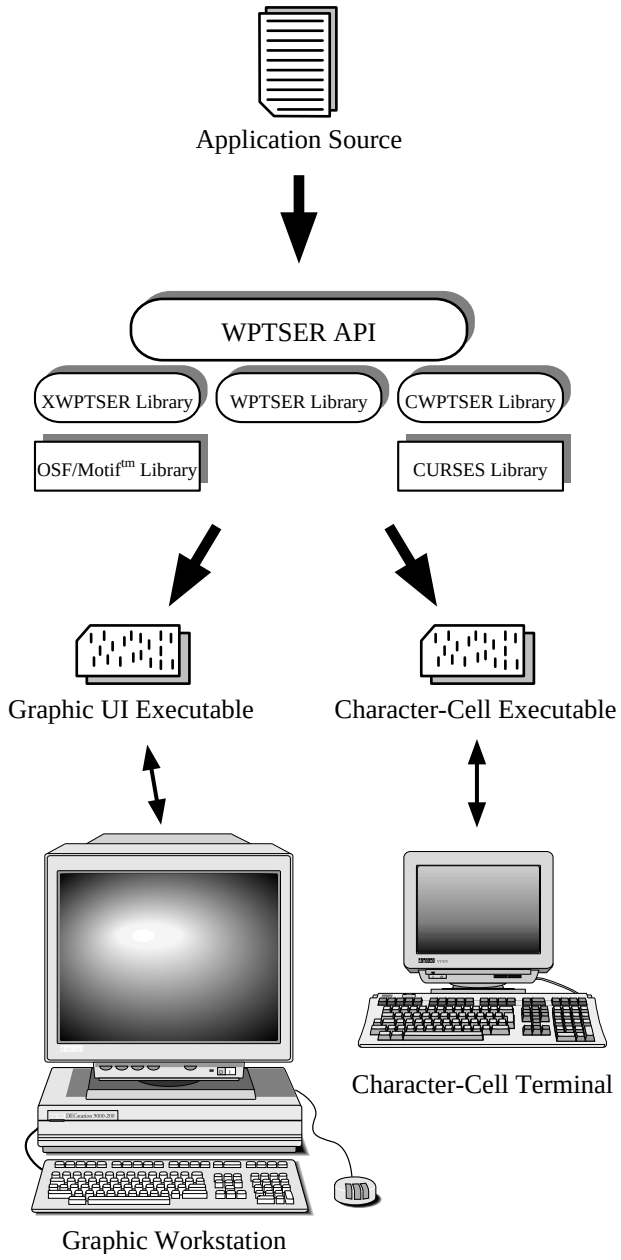
The application developer accesses *WPTSER* through its Application Programming Interface (*WPTSER API*) which is is a unique and consistent interface between the application and the platform-specific libraries. The *WPTSER API* provides high-level application procedures to create, modify, and manipulate the UI Object primitives, which, in turn, implement tools from the CURSES and OSF/Motif[tm] user interface libraries.

The *WPTSER API* is uniform to the application regardless of which of the supported implementation platforms is ultimately selected. With *WPTSER*, therefore, the selection of terminal platform becomes truly transparent to the

Erik E. Westbrook, Waiman Cheung, and Gilbert Babin. "An Object-Based User Interface for Manufacturing Information Integration." in Rensselaer's Third International Conference on Computer Integrated Manufacturing. Troy, New York, USA. May 1992. pp. 114-118.

application developer. Currently, two platforms are supported: OSF/Motif and CURSES.

**Figure 1: *WPTSER* implementation overview**




After designing an application, the developer can create an executable binary image to operate with the desired implementation platform simply by compiling with the proper compilation option flags. The application developer does not need to generate multiple software sources for the same task merely in order to support different types of terminal equipment and standards.

Indeed, the *WPTSER* architecture is extendible to support any other type of user interface standard, should such support be required; in that case, ancillary *WPTSER* primitives for the new platform would simply be added to the existing *WPTSER* libraries.

Naturally, the appearance of UI Objects on different types of systems will vary, and pointer operations will differ slightly from keyboard operations. For example, the OSF/Motif™ implementation supports both keyboard and pointer traversal, and the character-cell CURSES implementation provides object type distinction in some cases at traversal time). Nevertheless, the same objects will be at the same relative locations on all platforms, and will provide identical and consistent functions to the user and the application. Indeed, even the relative layout of the objects are specified by the application in device-independent screen coordinates, which *WPTSER* automatically adjusts on graphic systems for maximum consistency compatibility, and user intuition.

Finally, *WPTSER*'s overall portability and multi-platform characteristics would not be complete were it not for the extensive use of portable standards such as C, CURSES, and OSF/Motif™ as the building blocks of *WPTSER*, thus providing portability across many computing platforms as well.

## 4: UI Objects

UI Objects are the basic objects which comprise *WPTSER*'s set of user interface building blocks. *WPTSER* provides eight UI Objects, each of which has a unique function and a characteristic appearance. UI Objects provide services to the application through callbacks, which are functions that the application designates for execution upon receipt of a particular user event. Because of their varying function, some UI Objects have multiple callbacks while others have only one. Where possible, all objects comply with applicable standards of appearance and behavior (e.g. OSF/Motif™ compliance).

### 4.1: Dialog boxes

The first UI Object type is the dialog box. The dialog box is the parent container for most *WPTSER* objects, and provides the ability to define a particular user interface subset, presenting or removing that subset at any time. *WPTSER* mandates a strict hierarchical dialog stacking order and does not permit out-of-order input. This policy permits the application to rest assured that input will occur within the dialog context it expects, without requiring it to handle dialog context switching. (In the OSF/Motif environment, the dialog boxes are manifested as separate dialog shell windows which are considered transient children of the toplevel menu dialog).

Software activation of a defined dialog box brings it to the top of the stacking order and into full view. Activation also invokes an internal event loop, during which *WPTSER* processes all user input required by objects in the dialog box (through application-defined callbacks and input constraints) until the dialog box is closed or deactivated by the application.

### 4.2: Push buttons

Push buttons allow a user to trigger or activate simple operations, and are perhaps the most straightforward type of object supported by *WPTSER*.

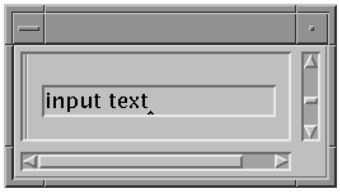
**Figure 2: Push button**


When the user activates the push button (with the mouse, pointer, or keyboard), the application callback function associated with the button is invoked.

### 4.3: Fields

Through the use of fields, the application can obtain alphanumeric input from the user. The application defines the field's input validity requirements using a regular expression.

**Figure 3: Field**

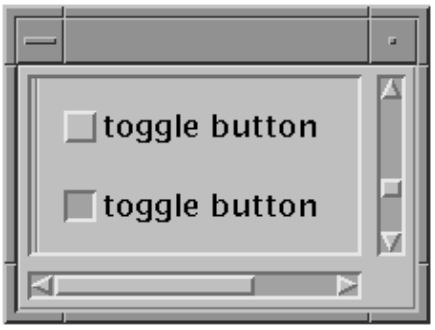



As the user attempts input through the field, the input is validated based on the regular expression, through the use of a deterministic finite automaton (DFA). The developer needs only to provide the regular expression; *WPTSER* automatically converts it internally to a minimized DFA based on known and proven algorithms [2]. Fields can be associated with activation callbacks if the application requires notification of the user's acknowledging completion of input.

### 4.4: Toggle buttons

The toggle button comes in two varieties, and provides a state or option selection mechanism. The first variety of toggle button has only one entry, and can be toggled on or off.

**Figure 4: Toggle buttons (off/on)**




An indicator (an asterisk in CURSES; a graphic "active" indicator in OSF/Motif) displays the current state of the toggle button. The second variety has many entries. Toggling the button in this case increments the state of the button to the next entry. In this case, *WPTSER* indicates the button's current state by changing the button's label to the current entry.

### 4.5: Lists

Often, an application will require the user to identify some subset of a set of items. For this purpose, *WPTSER* provides the list object.

**Figure 5: List**


The application defines a list of items (each of which may or may not be associated with a callback) as well as an optional master callback for the list and the number of entries selectable. Lists may allow one, some distinct

number, or all entries of the list to be selected. *WPTSER* handles all of the selection and deselection details, including rejection of excessive or invalid selections. On all platforms, list items exceeding the defined display size of the list area can be viewed and selected by scrolling.

### 4.6: Lines and arrows

Almost no dialog display is very meaningful or useful without some sort of item grouping or highlighting. Since intelligent device layout alone may not be sufficient to accomplish this task, *WPTSER* provides lines and arrows. Lines and arrows allow the user to set off, connect, or highlight certain areas of a dialog box for greater meaning to the user. Arrows in *WPTSER* are used specifically to connect toggle buttons and are defined by the application using the identifiers of the two toggle buttons to connect. The application defines the endpoints of lines in the same manner as locations of other devices.

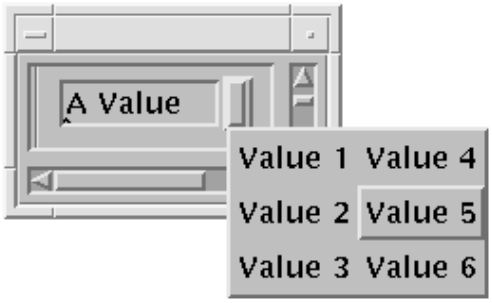
### 4.7: Menus and pulldowns

Like most graphic user interface applications, *WPTSER* provides a master "menu bar" for top-level application functions. Each item in the menu bar can be activated on its own, or may post a pull-down menu associated with that item. Through the use of callback functions, an application developer can quickly and easily generate a master command hierarchy to permit maximum user efficiency and, perhaps more importantly, more intuitive user understanding of the application's major objectives and principles.

### 4.8: Field lists

The field list is a specialty combination provided for convenience. In many situations, the application needs to provide the user with a set of likely options, but would also like to give the user the opportunity to make a selection not within the list.

**Figure 6: Field list**




The field list is ideal for such a situation. By offering both a field and a list in the same device, the field list offers the user this functionality in an intuitive and efficient manner. When inactive, the field list has the appearance of a simple field. When activated, however, the list appears and the user can either traverse and select from the list items or enter a selection manually.

## 5: A case study

A prototype Model-assisted Global Query System (MGQS) based on the direct knowledge approach [4] has been built using *WPTSER*. This prototype is a software system written in the C language and running on a Digital[tm] workstation. The implementation is an extended effort from Nogues' Global Query System [5]. The global query manager is fully integrated with the metadata manager (a prototype developed by Bouziane [3] which consists of a rule processor and a routine manager) to constitute the complete MGQS. It provides functions of global query formulation, global query optimization, local query translation, result integration, and on-line intelligence using a rule-based approach. The Computer Integrated Manufacturing Program at Rensselaer is employed as a test bed for the requirements, design, and implementation of the prototype.

The prototype MGQS user interface (figure 7) consists of four windows. The upper left window entitled *SPECIFY SCOPE FOR FORMULATION* is used for model traversal (browsing). A menu can be triggered for each subtitle (*Application*, *Subject*, and *Entity/Relationship*). Selected objects are displayed in the entry fields. The logic of using this window is that when an application (e.g., *ORDER_PROC*) is selected, only the subjects that are within this application will be listed under the *Subject* menu for selection. Similarly, when a subject (e.g., *ORDER*) is specified, only the entities/relationships that are within the subject will be listed for selection. The center window of the upper half of the screen, *FORMULATE QUERY*, operates similarly and is used for selecting data object into the global query.

The *Data Items* button triggers a list of data items within the boundary specified in the browsing window. When the second button, *Ent./Rel.* is activated, the entity/relationship specified in the browsing will be selected into the query. In this manner, a user can explicitly include an entity/relationship in a global query for semantic proposes even though no data item will be retrieved from it.

The progress of the formulation is displayed in the window on the lower half of the screen. Selection conditions for the query can be specified by moving the cursor to the proper row of the *QUERY FORMULATION IN PROCESS* window and key in an operator and the associ-

ate value/item of the condition.

The *Do Query* button in the upper right side of the screen will execute the formulated global query. The *Save MQL* button will translate the formulated global query into the syntax of the Metadatabase Query Language and then save it into a file. Lastly, the *QUIT* button will exit from the MGQS system.

## 6: Conclusion

*WPTSER*, then, is a new toolkit for development of graphical user interfaces. This toolkit breaks the three major barriers in creating user interfaces: Heterogeneity among terminal hardware (character-cell based or graphic workstation), the complexity of graphical packages, and input validation task. *WPTSER* enables the developer to create applications that will work on character-cell based terminal and graphic workstation by virtue of its standard application interface and the underlying implementation architecture. The application can be recompiled for different types of terminal hardware without the need for any changes whatsoever to the application source code. The use of *WPTSER* is simplified by the small number of interface devices, their consistency, and the minimization of the number of their formal parameters. These devices represent the different classes of interactions necessary to develop the user interface. Clearly, a large amount of the technical details associated with interfacing with users is therefore being handled by *WPTSER*. Since one of the most significant aspect of user interface development is input validation, *WPTSER* deals with most of the syntactical aspects of input validation. The developer must provide the syntax of the input only in the case of fields and field lists, and even in those cases, the syntax is defined using simple regular expressions, which not only provide a programming convenience to the developer, but also minimize the validation effort.

## 7: References

1. Babin, G., WPTSER: A Windowing Package for the Two-Stage Entity-Relationship Model, Tech. Report #CIMSI91TR220, Computer Integrated Manufacturing, Center for Manufacturing Productivity and Technology Transfer, Rensselaer Polytechnic Institute, Troy, N.Y., 1991
2. Aho, A.V. and J.D. Ullman, Principles of Compiler Design, Addison-Wesley, Reading, MA, 1977, pp. 73-103.
3. Bouziane, M., Metadata Modeling and Management, Ph.D. Thesis, Computer Science Department, Rensselaer Polytechnic Institute, Troy, N.Y., 1991.
4. Cheung, W. The Model-Assisted Global Query System, Ph.D. Thesis, Decision Sciences and Engineering Systems Department, Rensselaer Polytechnic Institute, Troy, N.Y., 1991.
5. Nogues, J., Global Query System, M.S. Thesis, Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY. 1990.

**Figure 7: Prototype MGQS user interface**