METHOD AND SYSTEM FOR ENCAPSULATING AN APPLICATION PROGRAM

Abstract: A system and method for encapsulating an application program (106) is described. The system uses a concise, high-level application encapsulation and description language syntax which uses a plurality of object descriptions to encapsulate a complete application program. The application encapsulation and description language syntax is designed to concisely, or orthogonally, provide a basis of fundamental data description objects that span the data problem space of business data access programming problems. The application encapsulation language syntax uses several types of objects (306, 324), which in combination encapsulate a wide range of application programs. Graphical user interface ("GUI objects") (308) encapsulate important user interface views and functions. A set of query objects (310) populates the GUI objects with data, a set of transaction objects (312) structure the GUI objects into linked views, and a set of control objects provide additional application features. An associated data source (108) may also be provided. These application description objects (318) are combined by an application shell (104) program that reads and parses one or more specification or configurations files written in the object description syntax and creates a fully-functional application program. This application program may be executed to allow access to, and manipulation of the associated data source.
Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.
METHOD AND SYSTEM FOR ENCAPSULATING AN APPLICATION PROGRAM

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

RELATED APPLICATIONS

This application is related to commonly owned application entitled METHOD AND SYSTEM FOR AUTOMATICALLY GENERATING AN APPLICATION PROGRAM BASED UPON DATA CHARACTERISTICS, filed October 22, 1999, attorney docket no. 3889/3, which is hereby incorporated by reference into this application.

BACKGROUND OF THE INVENTION

This invention relates to techniques for generating application programs.

More particularly, the invention relates to a system and method for using a set of objects to encapsulate, describe and generate fully-functional application programs.

Commercial application programs are generally difficult to create and maintain. Building commercial applications usually requires large financial and human resource investments. In addition, many thousands of lines of computer code are typically needed to produce full-featured applications, making modifications and enhancements difficult, and making simple coding errors problematic to locate and eliminate. Programmers have endless programming choices and often get mired in design details, frequently producing non-standard applications that are difficult to use. Although programmers typically spend substantial time on user interface development, content data is still difficult to effectively access, navigate, or manipulate, and many user interfaces are non-standard. This is partly because application programs, and particularly application program interfaces, are often not well matched to the underlying content data that users wish to view or manipulate.

Systems that encapsulate and describe computer programs are known those skilled in the art. For example, high-level computer programming languages, such as C++ or Java, can be used to describe an application program. However, existing systems may be characterized as requiring relatively extensive sets of instructions to create application programs with relatively low degrees of functionality and interoperability. This is largely
because high-level computer programming languages are designed to solve general problems, in what may be called the 'total problem space' of all possible computing tasks and problems. The instant invention does not treat the total problem space, but instead provides complete solutions to what may be called the 'data problem space' of all business data computing tasks and problems. Thus, the instant invention provides a concise, orthogonal set of tools that span the data problem space, and provide efficient, effective solutions to tasks and problems in the data problem space.

There is thus a need for methods and systems for more quickly and easily generating application programs with substantial functionality.

SUMMARY OF THE INVENTION

It is an object of the present invention to solve the problems described above associated with existing approaches to application program generation, and related problems of data access.

It is another object of the present invention to encapsulate an application program using a concise description syntax that maximizes the features and functionality of the application program.

It is another object of the present invention to provide a concise set of tools for solving business data problems in the data problem space.

It is another object of the present invention to provide an encapsulation syntax that may be stored or transmitted, that may be used by different application shell programs, and that produces application programs which run under different operating systems or user environments.

The present invention provides a system and method for encapsulating an application program. The invention provides a concise, high-level application encapsulation and description language syntax, described in detail hereinbelow. The application encapsulation and description language syntax uses a plurality of object descriptions to encapsulate a complete application program. The application encapsulation and description language syntax is designed to concisely, or orthogonally, provide a basis of fundamental data description objects that span the data problem space of business data access programming problems. The application encapsulation language syntax provides a significant improvement over existing techniques in terms of the number of instructions required and the time required to create a complete application program.
The application encapsulation language syntax uses several types of objects, which in combination encapsulate a wide range of application programs. Graphical user interface (‘GUI objects’) encapsulate important user interface views and functions. A set of query objects populates the GUI objects with data, a set of transaction objects structure the GUI objects into linked views, and a set of control objects provide additional application features. An associated data source may also be provided. These application description objects are combined by an application shell program that reads and parses one or more specification or configurations files written in the object description syntax and creates a fully-functional application program. This application program may be executed to allow access to, and manipulation of, the associated data source.

The objects described hereinabove generally take three different forms: application object descriptions, application level objects, and application specific objects. The application level objects provide preprogrammed generic structure and functionality for various aspects of an application program, and include GUI objects, transaction objects and others, all as described herein. The application object descriptions are programming language like statements that describe the various component objects. The application object descriptions, when read and parsed, and used to create application specific objects, which are active versions or instances of the application level objects. These application specific objects are then populated with content data when the application program is run and are assembled to create the application program.

An application program may be generated by an application shell, data centered browser, or other application generation software based upon a set of application object descriptions stored in one or more application specification files. The application shell program initially parses the application specification file to retrieve the object specifications or characteristics including an application object, one or more graphical user interface objects and one or more transactions, each transaction defining a screen layout. These object descriptions are used to generate application specific objects by applying the characteristics retrieved from the application specification file to the application level objects stored in the application specification file. Queries associated with each GUI object are executed against a specified data source to populate viewing screens that are used to build the application program, thereby allowing access to and manipulation of the data source.
The group of GUI objects generally consists of a calendar view object, a grid view object, a card view object, a tree view object, a drag & drop list object, a report view object, a graph view object, a drop down list object, and a detail view object. Further GUI objects may be developed by users or programmers and may be transparently integrated with the application encapsulation system.

The encapsulated application program, as embodied by an application specification file, may be retrieved or transmitted via a telecommunications connection or the Internet. Thus, the encapsulated program forms a basis for application program transmission among many geographically remote computers, which may be of different makes and models, running different operating systems and application environments. The encapsulated program may also be stored on virtually any type of computer readable media as a concise and cost effective way to distribute content data. For example, a encapsulated program may be stored and distributed on diskettes or CD-ROMs.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated in the figures of the accompanying drawings which are meant to be exemplary and not limiting, in which like references are intended to refer to like or corresponding parts, and in which:

Figure 1 is a block diagram of an exemplary application program encapsulation system in accordance with one embodiment of the present invention;

Figure 2 is a functional block diagram of the computing apparatus of Figure 1 that encapsulates computer applications;

Figure 3 shows the flow of data in an exemplary application program encapsulation system;

Figure 4 shows a schematic overview of an exemplary process for application level object generation and processing;

Figure 5 shows exemplary application level objects and their interrelationship in accord with an embodiment of the present invention;

Figure 6a is a schematic representation of the specification syntax for an application object in accord with an embodiment of the present invention;

Figure 6b is an example description of the specification syntax to create an application object in accord with an embodiment of the present invention;

Figure 7a is a schematic representation of the specification syntax for a
background object in accord with an embodiment of the present invention;

Figure 7b is an example description of the specification syntax to create a background object in accord with an embodiment of the present invention;

Figure 8a is a schematic representation of the specification syntax for a color object in accord with an embodiment of the present invention;

Figure 8b is an example description of the specification syntax to create a color object in accord with an embodiment of the present invention;

Figure 9a is a schematic representation of the specification syntax for an execute command object in accord with an embodiment of the present invention;

Figure 9b is an example description of the specification syntax to create an execute command object in accord with an embodiment of the present invention;

Figure 10a is a schematic representation of the specification syntax for an execute query object in accord with an embodiment of the present invention;

Figure 10b is an example description of the specification syntax to create an execute query object in accord with an embodiment of the present invention;

Figure 11a is a schematic representation of the specification syntax for an export object in accord with an embodiment of the present invention;

Figure 11b is an example description of the specification syntax to create an export object in accord with an embodiment of the present invention;

Figure 12a is a schematic representation of the specification syntax for a field object in accord with an embodiment of the present invention;

Figure 12b is an example description of the specification syntax to create a field object in accord with an embodiment of the present invention;

Figure 13a is a schematic representation of the specification syntax for a foldertab object in accord with an embodiment of the present invention;

Figure 13b is an example description of the specification syntax to create a foldertab object in accord with an embodiment of the present invention;

Figure 14a is a schematic representation of the specification syntax for a folder object in accord with an embodiment of the present invention;

Figure 14b is an example description of the specification syntax to create a folder object in accord with an embodiment of the present invention;

Figure 15a is a schematic representation of the specification syntax for an
import object in accord with an embodiment of the present invention;

Figure 15b is an example description of the specification syntax to create an import object in accord with an embodiment of the present invention;

Figure 16a is a schematic representation of the specification syntax for an outbar object in accord with an embodiment of the present invention;

Figure 16b is an example description of the specification syntax to create an outbar object in accord with an embodiment of the present invention;

Figure 17a is a schematic representation of the specification syntax for a query object in accord with an embodiment of the present invention;

Figure 17b is an example description of the specification syntax to create a query object in accord with an embodiment of the present invention;

Figure 18a is a schematic representation of the specification syntax for a startup object in accord with an embodiment of the present invention;

Figure 18b is an example description of the specification syntax to create a startup object in accord with an embodiment of the present invention;

Figure 19a is a schematic representation of the specification syntax for a table object in accord with an embodiment of the present invention;

Figure 19b is an example description of the specification syntax to create a table object in accord with an embodiment of the present invention;

Figure 20a is a schematic representation of the specification syntax for a transaction object in accord with an embodiment of the present invention;

Figure 20b is an example description of the specification syntax to create a transaction object in accord with an embodiment of the present invention;

Figure 21a is a schematic representation of the specification syntax for a view object in accord with an embodiment of the present invention;

Figure 21b is an example description of the specification syntax to create a view object in accord with an embodiment of the present invention;

Figure 22a is a schematic representation of the specification syntax for a viewfield object in accord with an embodiment of the present invention;

Figure 22b is an example description of the specification syntax to create a viewfield object in accord with an embodiment of the present invention;

Figure 23 is a flowchart describing a use of the program encapsulation in
accord with an embodiment of the present invention;

Figure 24 is a flowchart describing a process of generating a functional
application program from the encapsulated application program in accord with an
embodiment of the present invention;

Figure 25 is a flowchart describing a process of generating a specific data view
in accord with an embodiment of the present invention;

Figure 26 is a flowchart describing a exit sequence process for the generated
program in accord with an embodiment of the present invention;

Figure 27 is a diagram describing an example scenario where two client
workstations request and receive encapsulated application programs from a common server in
accord with the present invention; and,

Figures 28-32 are exemplary screen displays of an integrated application
program generated in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

As a general overview, the present invention is a system and method of
encapsulating an application program. The application program encapsulation is a concise,
high-level representation of a fully-functional application program from which the application
program itself can be generated. The application program encapsulation system and method
uses a set of application program objects and descriptions to combine formatting and control
information to encapsulate and describe fully-functional application programs.

The application program encapsulation system and method includes a set of
application encapsulation objects, also called application level objects or application object
descriptions. These descriptions may be generated by a computer program or may be hand-
coded using an application description language. A series of these application descriptions,
are typically generated and stored in a file or files, called an application specification file. In
the preferred embodiment, several files contain the application descriptions. The application
specification file contains descriptions for several types of application level objects,
including: GUI objects; query objects; transaction objects, and activation or control objects.
The GUI objects describe a specific viewing format or user interface specification for a set of
data. The query objects, which are each associated with a GUI object, populate instances of
the GUI objects with data to create interactive data views that a user can access and modify.
A set of associated transaction objects links the GUI objects into coherent interface structures.
The activation objects provide global controls for the generation of the application program as well as other information.

An application program shell, which may be referred to a data centered browser program, receives the GUI objects and uses the transaction objects and other objects to create views of the GUI objects, which are populated with data by executing the associated queries on a specified data source. The user of the resulting program can view, navigate and modify content data from a data source via the GUI objects.

Embodiments of the present invention will now be described in detail with reference to the accompanying figures. Figure 1 shows a block diagram of the system and method for encapsulating an application program in accordance with one embodiment of the present invention. Illustrated in Figure 1 is an application specification file 102, which contains a full set of application description statements that are written in the application encapsulation language syntax described in greater detail below. An application shell program 104 reads and parses the application specification file 102. The application shell program 104 references a set of stored application level objects and applies the descriptions parsed from the specification file 102 to create a set of application specific objects 112. The application specific objects 112 are assembled into an application program 106. The application specific objects 112 are populated with a data set that is read from a data source 108. The data source 108 may potentially originate from one or more files, other application programs, realtime data feeds or may be derived from other data sources, without limitation, as known to those skilled in the art.

Figure 2 shows a block diagram of an apparatus used to generate an application program 106 from an application program encapsulation. The apparatus has a central processor unit (CPU) 204 that is coupled to and accesses memory devices 202 and input / output devices 206. The memory devices 202 may include random access memory, disk storage and other forms of volatile or non-volatile storage. The input / output devices 206 may include user interface devices, network interfaces, and other computing peripherals. These hardware elements may be conventional elements such as in a mainframe computer, personal computer, laptop computer, handheld computer, or personal digital assistant, as known to those skilled in the art, and are controlled by an operating system and other conventional software.
A set of parsing rules 214 recognizes application level object descriptions read from an application specification file. The application shell program 104 uses application level objects 110 and applies the descriptions from the application specification file parsed according to a set of rules 214 to create application specific objects. The rules 214 may be hard coded into the application shell program or may originate from direct user selection or from preferences files. Once created, the application specific objects are populated with data from the data source 108 to create the final application program 106.

With reference to Figure 3, a diagram showing the flow of data in the application program encapsulation system and method is presented. The application program encapsulation specification file 102 contains one or more application level objects 306, and a plurality of activation objects 324. Each of the application level objects 306 is a concise, high-level representation of a portion of an application program that includes GUI objects 308, query objects 310, and transaction objects 312. Each GUI object 308 is supported by a corresponding query object 310 that is used to populate the corresponding GUI object instance 322 with data. The transaction objects 312 structure the GUI objects 308 into interactive display views. The set of activation objects 324 included in the encapsulated application program 302 provide overall structure and interface elements for the generated application program, such as the title of the generated application program, folders, menus, outbars and other standard user interface features and controls.

The generated application program 106 is created from the application program encapsulation 302. In the generated application program 106, the application level objects 308 are instanced as application specific objects 318. Each application level object 318 combines a transaction object instance 320 with one or more GUI object instances 322. For each GUI object instance 322 to be displayed, the application program generator interprets a GUI object 308 and populates a corresponding GUI object instance 322 with data using the query 310 associated with the GUI object 308. A GUI object instance 322 is structured into a specific display by a transaction object 320 to present content data. Thus, a GUI object 308 is a general schema, structure and description for encapsulating data and program information while a GUI object instance 322 is fully functional application program interface element that enables direct user interaction with a data set.

There are several GUI object types 308 including: grid objects, card objects, detail objects, tree view objects, calendar objects, graphic objects, drag & drop objects, drop
down list objects, home page objects, and report objects. Furthermore, additional GUI object
types 308 may be defined at the user or system level to customize the application generator to
specific data environments. Examples of specific GUI object types 308 are described
hereinbelow.

With reference to Figure 4, an overview of the process of creating and using
the application program encapsulation system and method are presented. In step 402, the
application level objects are created and stored, such as in a library. For commercial
embodiments, a library of application level objects are distributed along with the application
shell program and a designated description syntax to facilitate easy program generation. A
data source may also be created at this stage in one embodiment, or in an alternate
embodiment the data source can be created, retrieved or accessed when the generated
application program is run. In step 404, application descriptions are written and are stored in
an application specification file. The application specification file is later read and parsed,
step 406, by a shell program, data centered browser, or other program. Once read and parsed,
a set of application specific objects is generated according to the application object
descriptions stored in the specification file, step 408. In step 410, an application program is
generated by assembling the application specific objects into the complete application
program. The application specific objects are populated with data from the data source to
create data displays when a specific view is selected by a user at runtime.

Figure 5, shows a set of application level objects provided in one embodiment
of the invention in generating a data management application program. An Application
Object 500 defines the top level characteristics of the application program. A Startup Object
501 is used to specify default behavior at application startup including which GUI objects are
hidden from view. A FolderTab Object 502 defines a high level grouping under which Folder
Object 504 definitions can be made. A Background Object 512 defines default backgrounds
colors for a wide variety of view types, such as calendars and reports. A Color Object 514
defines a default color that is available to the application.

An ExecuteCommand Object 550 executes the specified system command,
such as a file delete operation. An ExecuteQuery Object 536 will execute a specified query.
An ExportObject 548 executes the specified query and writes a record for each row of the
returned record set. A Field Object 558 defines the default characteristics of a database field
that is used in the application. An Import Object 534 imports a specified file into a specified
table. An OutBar shortcut 508 or Outbar group 510 is used to have an icon appear as a
selection on the Outbar also called a shortcut menu, located on the leftmost portion of the
screen (see, e.g. - in Fig. 28). A Query Object 518 is used to define a SQL query to execute.
A Table Object 520 defines the default characteristics of a database table that is used in the
application. A Transaction Object 506, one of the most structurally important objects in the
application program encapsulation method, is used to define the screen layout. A View
Object 532, also referred to herein as a GUI object, completes the structural setup specified
by the Transaction Object. View objects may be of several types, and new view objects may
potentially be added to extend the system. The view types are: a grid view 524, a detail view
526, a card view 528, a calendar view 530, an image view 546, a drag & drop list 544, a
graph view 542, a report view 540, a tree view 538, and a home page view 522. Exemplary
screen displays of these various views are shown in Figs. 28-32.

The objects contain declarations and methods used to accomplish the tasks
outlined above and to generate aspects of an application program in accordance with
processes described below. The objects contain a number of variables, values from which are
supplied in the specification file or files according to the predefined syntax this syntax, for
variables and their functions within each object, and example specification or description
statements for each object, are set forth below with reference in Figs. 6a-22b.

With reference to Figure 6a, an encapsulation syntax 602 for the Application
Object 552 is described according to the preferred embodiment. The Application Object 552
defines the top level characteristics of the application program. This includes the name of the
application as well as the details on the source of the data.

An Application Object contains the following additional information fields.
The field ‘Object = [APP]’ identifies the beginning of an Application Object definition. The
field ‘AppName = [appname]’ identifies the name of the application to be generated. A
statement of the form ‘Name = [name]’ determines the value that is displayed in the Title Bar
of the Application when running. The field, ‘DBName = [dbname]’, specifies the data source
that the application is to use when running. If the name cannot be found, a new datasource
is automatically created. The field, ‘CompactFrequency = [nnn]’ specifies how often to
compact and repair a database when exiting the application. The field, ‘DatabaseSignon =
[username | databaseusername | NONE]’ specifies the application to display the Database
User Name field on the sign-on screen. The field, ‘DatabasePassword = [YES | NO]’,
specifies that a database password field exists on the sign on screen and the password must be entered. The field, ‘DBFileName = [dbname]’, specifies a database file name. If the DBName requires a specific file, usually in the case of non-server-based database technologies, this is the file name and location. If the DBFileName does not include a path then the system will use the same directory as the encapsulation file. The field, ‘NewPassword = [YES | NO]’, specifies whether or not a new password box is displayed. The field, ‘Registration = [YES | NO]’, specifies to display the Registration number field. The registration number is used to track registered users of a given application. The field, ‘RegistrationPrefix = [xxxx]’, specifies the prefix to use to compare to what the user enters. The field, ‘RegistrationMessage = [xxxx]’, specifies the text to display if the software is not registered. The field, ‘RegistrationURL = [xxxx]’, specifies the URL to display so a user can click to register the software. The field, ‘SaveDatabasePasswordOption = [YES | NO]’, specifies whether to show a ‘Save Database Password’ checkbox on the sign-on screen. The field, ‘SavePasswordOption = [YES | NO]’, specifies whether to show a ‘Save Password’ checkbox on the sign-on screen. The field, ‘SplashFile = [splashfilename]’, specifies an image file to display when loading the application. The field, ‘SplashText = [splashtext]’, indicates that if there is no SplashFile, then display the SplashText in black letters on the Background specified in Color. The field, ‘UserDropDown = [YES | NO]’, specifies whether to create a list of Users from the UserPreferences table. The field, ‘Version = [version]’, specifies the version number to display in the application’s about box. The field, ‘Versiondate = [versiondate]’, specifies a creation date to assign to the application, which is displayed in the about box.

Figure 6b presents an example of the specification syntax for a specific Application Object 644. It defines version 1.1 of the specification syntax for an application called ‘MyApp’, with a title ‘My Sample Application’, uses an DBName entitled ‘MyApp’, and a database file ‘MYAPP.MDB’.

With reference to Figure 7a, an encapsulation syntax 702 for the Background Object 512 is described according to the preferred embodiment. The Background Object 512 defines default backgrounds colors for a wide variety of view types, such as calendars and reports. This is used to define a background color that is used as a default whenever that type is requested by the application. These colors are defaults and can be overridden when required.
A Background Object contains the following additional information fields. The field ‘Object = [BACKGROUND]’, identifies the beginning of the specification syntax for a Background Object definition. The field, ‘AppName = [appname]’, specifies which application the syntax applies to. The field, ‘SubObject1 = [report | calendar...]’, specifies the primary object type to apply the default background color to, such as a report or calendar. The field, ‘Color = [red | blue...]’, specifies the desired color to display. The color should be a value already defined in the Color Object. The field, ‘SubObject2 = [title | heading...]’, specifies the secondary object type to apply the default background color to, such as a title within a report. The field, ‘Name = [foldertabobject]’, specifies a FolderTab object. If Default is specified in SubObject1 then ‘Name’ specifies the FolderTab object to apply the default color to.

Figure 7b presents an example of the specification syntax for a specific Background Object 716. It defines a background color of ‘SkyBlue’ for the ‘DayHeadingFocusDay’ section of a ‘Calendar’ GUI object.

With reference to Figure 8a, an encapsulation syntax 802 for the Color Object is described according to the preferred embodiment of the present invention. The Color Object defines a default color that is available to the application. The field, ‘Object = [COLOR]’, specifies the beginning of a Color Object definition. The field, ‘AppName = [appname]’, specifies which application the syntax applies to, which is already defined in the AppObject. The field, ‘Name = [colorname]’, specifies the name of the color, such as ‘Red’. The field, ‘RGB = [r,g,b]’, specifies red, green, and blue components of a specific color value.

Figure 8b shows a color object 812 that defines a color ‘Red’ that is available to all applications.

With reference to Figure 9a, the ExecuteCommand Object 550 encapsulation syntax 902 is described according to the preferred embodiment. The ExecuteCommand executes a specified command. The field, ‘Object = [EXECUTECOMMAND]’, identifies the beginning of a ExecuteCommand Object definition. The field, ‘Command = [commandtext]’ specifies the text of the command to execute. These are frequently file level commands, like deleting or copying files.

Figure 9b shows an ExecuteCommand Object description 908 that performs a copy file function.
With reference to Figure 10a, an encapsulation syntax 1002 for the ExecuteQuery Object 536 is described according to the preferred embodiment. An ExecuteQuery object will execute a specified query. The field, ‘Object = [EXECUTEQUERY]’, identifies the beginning of a ExecuteQuery Object definition. The field, ‘Query = [queryname]’, specifies the query to execute.

Figure 10b shows a query command description 1008 that updates a customer table.

With reference to Figure 11a, an encapsulation syntax 1102 for the ExportObject 548 is described according to the preferred embodiment. The ExportObject executes a specified query and writes a record for each row of a record set that is returned by the query. The field, ‘Object = [EXPORT]’, identifies the beginning of a ExportObject definition. The field, ‘Query = [queryname]’, specifies the name of the Query Object to execute. The field, ‘FileName = [drivepathfilename]’, specifies the name of the file to export to. The field, ‘FileType = [CSV | ZGC]’, specifies the type of file to create. For example, ‘CSV’ is an industry standard file containing comma separated values. In general, any file type may be added.

Figure 11b shows a export object description 1110 that exports a CSV file using the Customers Query Object.

With reference to Figure 12a, the encapsulation syntax 1202 for a Field Object 558 is described according to the preferred embodiment of the present invention. The Field Object defines default characteristics of a database field used in the application program. This can be used to specify various attributes, such as color, size, or type. The field, ‘Object = [FIELD]’, identifies the beginning of a FieldObject definition. The field, ‘AppName = [appname]’, specifies the application to which the field applies. The field, ‘Name = [fieldname]’, specifies the name of the database field, for example, ‘Start Date’. The field, ‘Query = [queryname]’, specifies the name of the query to execute when displaying this field controls, if a Control type of ComboBox is specified. The query should be as follows: select value, valuename from table The first column value specifies the value to be stored on the database, while the second column valuename specifies the name to display when viewing the field. If no ‘Order By’ clause is specified, the Query results will automatically be ordered in the same sequence as the non-hidden values are shown. The field, ‘BlankWhenZero = [Yes | No]’, is set to ‘Yes’ to display zeros as blanks or ‘No’ to show zeros. The field, ‘Calculate =
defines the expression to execute when any Field specified in the expression has changed, such as addition, subtraction, multiplication and division. Operations are performed left to right. Here, the FieldName specifies a field from the current record to include in the expression; Prev.FieldName indicates that a field from the prior row in a grid should be used in the calculation; First.FieldName indicates that a field from the first row should be used in the calculation; Last.FieldName indicates that a field from the last row should be used in the calculation; ‘Now()’ indicates that the current Date & Time should be used in the calculation; and, ColorCondition = [< | > | =]’, specifies to change the color of the text in the field when the condition is met. The field, ‘ColorConditionColor = [red | blue...]’, specifies the color to use when the ColorCondition is met. The color should be a value previously defined in the Color Object. The field, ‘Control = [COMBOBOX | DATE...]’, specifies the type of control to drop down when working in a field. Possible values are: CALCULATOR, COMBOBOX, CHECKBOX, DATE, DATE/TIME, MAILTO, and URL, and this list of values may be extended.

Continuing with reference to Figure 12a, the field, ‘ControlName = [controlname]’, specifies the name of the CustomControl Object. This is used to specify a specific custom control. The field, ‘DefaultValue = [Number | Now()...]’, specifies a default value to place in the field when a field is first initialized. For example, the keyword ‘Number’ specifies a numeric value like 10. The field, ‘Name = [FieldName]’, specifies a field from the current record to include in the expression; Prev.FieldName indicates that a field from the prior row in a grid should be used in the calculation; First.FieldName indicates that a field from the first row in a grid should be used in the calculation; Last.FieldName indicates that a field from the last row in a grid should be used in the calculation; ‘Now()’ indicates that the current Date & Time should be used in the calculation. The field, ‘DefaultColor = [red | blue...]’, specifies the color that is seen when the field is displayed. The color should be a value already defined in the Color Object. The field, ‘Format = [FLOAT | COMMA...]’, specifies the format that is applied to the field, where the value FLOAT indicates a floating point number; the value DOLLARS indicates a currency field; the value COMMA specifies a numeric field with commas; and, the value PERCENT specifies a numeric percent field. The field, ‘Hide = [YES | NO]’, hides the field from view. The field, ‘Justification = [LEFT | RIGHT | CENTER]’, specifies how to position the text within a cell. The field, ‘Places = [number]’, specifies the number of decimal places in a
numeric field. The field, ‘Size = [number]’, specifies the size in characters to make a field.
The field, ‘Type = [LONG | SHORT...]’, specifies the numeric type of the field. Valid
values are: DOUBLE, SHORT, LONG, INT, DATE, DATETIME, TIMESTAMP, and this
list of values may be extended.

Figure 12b describes a Field Object 1238 that defines a Date/Time field so that
a Date/Time custom calendar control is displayed.

With reference to Figure 13a, the encapsulation syntax 1302 for a FolderTab
Object is described according to the preferred embodiment of the present invention. A
FolderTab Object defines a high level grouping under which Folder Object definitions can be
made. FolderTab Objects, when defined automatically, become high level groupings for the
‘Go’ Menu, Outbar, Tree view and Home page. The field ‘Object = [FOLDERTAB]’,
identifies the beginning of a FolderTab Object definition. The field, ‘AppName =
[appname]’, specifies to which application the syntax applies. The field, ‘Name =
[foldertabname]’, specifies the name of the FolderTab Object. This name is used when
defining Folder Objects. The field, ‘Sequence = [number]’, identifies the order in which
FolderTab items are displayed.

Figure 13b shows a Foldertab Object description 1312 that defines a high level
group called ‘time’.

With reference to Figure 14a, the encapsulation syntax 1402 for a Folder
Object 504 is described according to the preferred embodiment of the present invention. A
Folder object defines a collection of Transaction Objects or group headings under a
FolderTab Object. This is used to define transactions to execute when items from the ‘Go’
menu, Outbar, Tree View or Home Page are selected. The field ‘Object = [FOLDER]’,
identifies the beginning of a Folder Object definition. The field, ‘AppName = [appname]’,
specifies which application the syntax applies. The field, ‘Name = [foldername]’, specifies
the name of the Folder Object. This name is displayed on the ‘Go’ menu, Outbar, Tree View
or Home Page. It may be a Heading, such as on a home page, or a selectable item. The field,
‘TabName = [foldertabname]’, specifies the FolderTab Object name to associate this Folder
Object with. The FolderTab Object must have been previously defined. The field, ‘Sequence
= [number]’, identifies the order in which to display Folder items. The field, ‘Parent =
[foldername]’, specifies the foldername of the Parent. If no value is supplied, this Folder
Object is a parent. If a foldername is specified, this Folder Object is a child, which is
displayed under a parent. The field, ‘Transaction = [transactionname]’, specifies the transaction to execute when the user selects a Folder Object.

Figure 14b describes a folder object 1418 that executes a Transaction called ‘Time Home Page’ when the ‘Time Home Page’ Folder Object is selected from a ‘Go’ menu, Outbar, Tree View or Home Page.

With reference to Figure 15a, the encapsulation syntax 1502 for an Import Object 534 is described according to the preferred embodiment of the present invention. The Import Object imports a specified file into a specified table. If the table does not exist, it is created. The field ‘Object = [IMPORT]’, identifies the beginning of an Import Object definition. The field, ‘TableName = [table]’, specifies the name of the database table to import data into. If the table does not exist, it is created. The field, ‘FileName = [drivepathfilename]’, specifies the name of the file to import data from. The field, ‘FileType = [CSV | ZGC]’, specifies the type of file import. This list may be extended to include additional data file types.

Figure 15b shows an import object description 1512 that exports a CSV file using a Query Object.

With reference to Figure 16a, the encapsulation syntax 1602 for an OutBar Object or Outbar Shortcut Object is described according to the preferred embodiment of the present invention. The Outbar Object is used to have an icon appear as a selection on the Outbar or shortcut menu, which is located on the leftmost portion of the screen. When a user selects an OutBar Object icon, and an associated Transaction Object is also executed. The field ‘Object = [OUTBAR]’, identifies the beginning of an OutBar Object definition. The field, ‘AppName = [appname]’, specifies which application the syntax applies. The field, ‘Name = [outbar]’, specifies the name of the Folder Object. This name is displayed on the ‘Go’ menu, Outbar, Tree View or Home Page. It may be a Heading, for example on a home page, or another type of selectable item. The field, ‘TabName = [foldertabname]’, specifies the FolderTab Object name to associate this Folder Object with. The FolderTab Object must have been previously defined. The field, ‘Sequence = [number], identifies the order in which items are displayed. The field, ‘Parent = [foldername]’, specifies the foldername of the Parent. If this field is blank then this Folder Object is a parent. If foldername is specified then this Folder Object is a child. The field, ‘Transaction = [transactionname]’, specifies the Transaction to execute when the user selects a Folder Object.
Figure 16b describes an Outbar Object 1618 that executes a Transaction called 'Time Home Page' when the 'Time Home Page' Outbar Object is selected from an Outbar view.

With reference to Figure 17a, the encapsulation syntax 1702 for a Query Object is described according to the preferred embodiment of the present invention. The Query Object is used to define a SQL query to execute. The field, 'Object = [QUERY]', identifies the beginning of a Query Object definition. The field, 'AppName = [appname]', specifies which application the syntax applies to. The field, 'Name = [queryname]', specifies the name of the Query Object. The field, 'DetailTable = [tablename]', specifies the name of the primary database table used in the Query. The field, 'SQL = [sqlquery]', specifies the SQL syntax that is executed. The query is preprocessed and then passed to the database access driver that is specified by the DBName parameter of the AppObject. The query should be in the format supported by both the driver and back end database. Queries are used to fill Views, run Action Queries, load controls and run initialization queries. If the order by clause is omitted from Select queries, the sort is in the order of the first five unhidden fields. The following special syntax is supported: %V[FROM] replaces this variable with the 'from' date that is the default, initially determined by the Value parameter on the View Object, before the query; %V[TO] replaces this variable with the 'to' date that is the default, initially determined by the Value parameter on the View Object, before the query; %F[fieldname] is parsed and will replace this parameter with the contents of the field name on the current row. An Action query points to this query to perform cascading deletes. This query uses the %F[fieldname] parameter to use the current row value as part of the 'where' clause. The %F[fieldname] is enclosed in quotes because the Item field supports character data.

Figure 17b shows an Query Object description 1714 with an embedded SQL command.

With reference to figure 18a, the encapsulation syntax 1802 for a Startup Object is described according to the preferred embodiment of the present invention. The Startup Object is used to specify which Views to hide at application startup. The field, 'Object = [STARTUP]', identifies the beginning of a Startup Object definition. The field, 'AppName = [appname]', specifies which application the syntax applies to. The field, 'SubObject1 = [folder | outbar...]', specifies the primary object type to apply the default behavior to. The field, 'SubObject2 = [default]', specifies the secondary object type to apply
the default behavior to. The field, ‘Sequence = [number]’, identifies the order in which to process Startup Objects. The field, ‘Hide = [YES | NO]’, hides the SubObject1 type from view.

Figure 18b shows a startup object description 1816 that hides the Folder View from displaying when the application starts up.

With reference to figure 19a, the encapsulation syntax 1902 for a Table Object is described according to the preferred embodiment of the present invention. The Table Object defines the default characteristics of a database table that is used in the application. The Table Object is used to indicate the primary database key and whether it is hidden. Also, Action Items can be created that will execute when certain events occur on the table, for example, deleting a row.

Continuing with reference to figure 19a, the field, ‘Object = [TABLE]’, identifies the beginning of a Table Object definition. The field, ‘AppName = [appname]’, specifies which application the syntax applies. The field, ‘Name = [fieldname]’, specifies the name of the database table, for example, ‘Customer’. The field, ‘SubObject1 = [define | Action]’, is set to ‘define’ if a table is being defined. ‘Action’ is specified if an action query is being declared. In the field ‘SubObject2 = [afterinsert | afterdelete…]’, the parameter afterinsert executes a specific query when any insert event occurs on a table; afterupdate executes a specific query when any update event occurs on a table, and afterdelete executes a specific query when any delete occurs on a table. The field, ‘Sequence = [number]’, identifies the order in which to process Action queries for a table. The field, ‘Query = [queryname]’, specifies the name of the Query Object to execute when a Action event is triggered. The field, ‘PrimaryKey = [fieldname]’, specifies the database field name that is used as the primary key for the table. Because primary keys may be hidden, the field ‘PrimaryKeyHidden = [YES | NO]’ can be used to hide the primary key field from view.

The field, ‘Recurring = [YES | NO]’, specifies that this table supports recurring processing. Recurring processing affects the Calendar view and will show activities that are applicable to a calendar. Recurring processing requires the following fields be defined on the table:

‘Recurring (yes/No)’ indicates the row is a recurring row; ‘Recurring Start Date (Date/Time)’ the date the recurring activity is to start; ‘Recurring End Date (Date/Time)’ the date the recurring activity is to end; ‘No Recurring End Date (yes/no)’ indicates that the recurring activity will never expire; ‘Frequency (text 15)’ how often the activity recurs (daily, weekly,
etc.); ‘Day Of Week (text 15)’ which day of the week (Monday, Tuesday, etc.); ‘Occurrence (text 15)’ when it occurs (every, first, second, third, fourth); ‘Day Of Month (text 15)’ the day of the month (1st, 2nd, 3rd, 4th, ...31st). The field, ‘Reminder = [YES | NO]’, specifies that this table supports reminder processing. Reminders will pop up when the application starts and whenever the setting for a reminder activity becomes valid, such as the 15 minute period prior to an appointment. Reminder processing requires the following fields be defined on the table: ‘Reminder (yes/No)’ indicates the row is a reminder row; ‘Remind In Advance (text 20)’ how many minutes in advance to remind (0 Minutes, 5 minutes, 10 minutes,...30 days); ‘Last Reminder (Date/Time)’ internal requirement; ‘Next Reminder (Date/Time)’ internal requirement. The fields, ‘ReminderFields = [fieldname1, fieldname2,...]’, specify which fields to display when a reminder pops up. The field, ‘ReminderTran = [transactionname]’, indicates which Transaction Object to open when the user clicks ‘Open Item’ on a reminder popup. This allows the user to drill into the item he is being altered to. In the field, ‘ReminderTranDDLValue = [ALL | ddlvalue]’, the parameter ‘ALL’ specifies not to limit any Drop Down List on the transaction or a specific ddlvalue. The field, ‘ReminderTranTabName = [tabname]’ indicates which ‘View’ to pre-select when drilling down to a transaction from a reminder popup.

Figure 19b shows an object description 1934 of a table with a hidden primary key.

With reference to Figure 20a, the encapsulation syntax 2002 for a Transaction Object 506 is described according to the preferred embodiment of the present invention. Transaction objects are one of the most structurally important objects in the application program encapsulation method. The Transaction Object is used to define the screen layout in an application. The Transaction Object defines how many screen sections to open, their position and how they coordinate. Applications can coordinate views so that selections in one window can trigger related display changes in another window. Transaction Objects are activated when a user selects a choice from a ‘Go’ Menu, Outbar, Home Page, Workspace Tab, folder view, and in other circumstances (these elements being shown and described with reference to Figs. 28-32). Transaction Objects are always defined in groups that have at least one screen section definition.

Continuing with reference to Figure 20a, the field ‘Object = [TRANSACTION]’, identifies the beginning of a Transaction Object definition. The field,
‘AppName = [appname]’, specifies which application the syntax applies. The field, ‘Name = [fieldname]’, specifies the name of the Transaction Object. This object is referred to by the Transaction parameter on the Folders or Outbar object. The field ‘SubObject1 = [define | TopView1...]’, specifies the primary object type to be defined. Valid parameters are: Define, which defines the basics of a Transaction; TopView1 which defines the left most top of screen; TopTab1 which is a tab control within a TopView1; TopView2 which defines the right of TopView1; TopTab2 which is a tab control within a TopView2; BottomView1 which defines the left most bottom of the screen; BottomTab1 which is a tab control within a BottomView1; BottomView2 which defines the right of BottomView1; BottomTab2 which is a tab control within a BottomView2; and, FarRightView which is always on the rightmost portion of the screen and is suited for Drag & Drop views. The field, ‘Sequence = [number]’, identifies the order in which to process TopTab1, TopTab2, BottomTab1 or BottomTab2 parameters. A drop down view showing all tabs listed under a tab heading is displayed in the order in which they are sequenced.

The field, ‘TabName = [tabname]’, specifies the title that is displayed on the horizontal or vertical tab. The field ‘Color = [red | blue...]’, specifies the color to use as the base color for this transaction object. The color should be a value previously defined in the Color Object. The field ‘FarRightPercent = [n]’, specifies the percentage of the screen workspace to allocate to the far right screen section. The field, ‘IconName = [iconname]’, specifies the icon to want use on the workspace tab when this transaction is open. The field, ‘IndependentPercent = [n]’, specifies the percentage of the screen workspace to allocate to the Independent splitter screen section. The field, ‘LeftPercent = [n]’, specifies the percentage of the screen workspace to allocate to the left screen section.

The field, ‘Orientation = [HORIZONTAL | VERTICAL]’, specifies whether the splitter will operate in a horizontal or vertical fashion. If vertical, the independent window is a full vertical window within the workspace. If horizontal, the independent window is a full horizontal window within the workspace. The field, ‘TopPercent = [n]’, specifies the percentage of the screen workspace to allocate to the top screen section.

The field, ‘ViewName = [viewname]’, specifies the name of the View Object, such as a calendar object, report object, or other object type. that is displayed in this screen section. The field, ‘ControlledByField1 = [fieldname]’, is used to coordinate different views in a transaction. It specifies the name of the database field from the controlling view that
controls this view. The view on which ControlledByField1 is specified will limit what is displayed in this view to items with the value of the ControlledByField1 passed from the controlling window. ControlledByField1 always has a related mirror ControllingField1. The field, ‘ControlledByField2 = [fieldname]’, is used to coordinate different views in a transaction. It specifies the name of the database field from the controlling view that controls this view. The view on which ControlledByField2 is specified will limit what is displayed in this view to items with the value of the ControlledByField2 passed from the controlling window. The field, ‘ControllingField1 = [fieldname]’, specifies the name of the primary database field that will control a subordinate window. The controlled view will limit what is displayed by this field value. The field, ‘ControlledByField2 = [fieldname]’, specifies the name of the primary database field that will control a subordinate window. The controlled view will limit what is displayed by this field value. The field, ‘ControlsView1 = [viewname]’, specifies the name of a view that this view controls. The field, ‘ControlsView2 = [viewname]’, specifies the name of a view that this view controls.

The field, ‘TabStyle = [T | D]’, specifies the type of tab to use. A Drop down tab will take the tab selections and create a drop down list in the order of the sequence numbers on the view. A Tab will create separate horizontal tabs that display selections listed in sequential order.

Figure 20b shows a transaction object description 2048 that creates a two section workspace, a top and bottom. In this example, the top view controls the display of the bottom view by CustomerID. The top view will point to a View Object that is a Grid View, while the bottom view will point to a View Object that is a Detail view. The ‘Name’ parameter on all entries below share the name ‘Customers’, which specifies they are part of the same transaction named ‘Customers’.

With reference to Figure 21a, a View Object or GUI object encapsulation syntax 2102 is described according to the preferred embodiment. The View Object completes a display setup created in a Transaction Object. A view object is filled by the results of a query and is displayed based on the characteristics of the view. The View Object is used to define the type of view that is displayed in a screen window. Views can be Reports, Grids, Trees, or other object types. Frequently, a Transaction is defined that specifies different views of the same data, like a Grid on the top and a Detail or Card view on the bottom. The
View Object is used to specify the characteristics of the different screen sections defined by a transaction.

Continuing with reference to Figure 21a, the field ‘Object = [View]’, identifies the beginning of a View Object definition. The field, ‘AppName = [appname]’, specifies which application the syntax applies. This should be a value already defined in the App Object. The field, ‘Name = [fieldname]’, specifies the name of the View Object. This object is referred to by the ViewName parameter on the Transaction Object. The field, ‘SubObject1 = [DEFINE | DATES | SECTION | ACTION | TOTALROW | FIELD]’, specifies the primary object type to define. Typically, at least one View Object entry is defined. Valid choices are: Define, which defines the basics of a View; Dates, which specify the default date range to use when first displaying a report or graph; Section is used to define a Report Section; Reports can have multiple sections of different types, like Detail Top and Grid bottom; Action is used to specify that an action can take place on the view; TotalRow is used to sum one or more columns in a Grid and display a Total Row. Event processing can be performed on Total rows. If the Query parameter is specified, then a query is executed whenever the total row changes. Field is used to override default characteristics for fields that just pertain to the current view. The field, ‘SubObject2 = [default | report…]’, specifies the type of view to define. Many different view types are supported, each one with a different purpose, look and feel.

Continuing with reference to Figure 21a, a Card is a non-editable list based the result set of a Query Object that is well suited for display in a ‘card’, for example, peoples names and addresses. In a card, a quick menu bar, the letters ‘a’ through ‘z’ are used to quickly locate an entry. Calendar is a non-editable data-based display based on input values from the results set of a Query Object. Only items that should be visible in a current date view are shown. Calendars are often coordinated with a Detail View with tabs. A DragDropList is a list of database rows from a query that can be dragged onto a specific view. Default is used when SubObject1 is set to DATES. Detail is a single row field list of a single database row and is often coordinated with another view, like a Grid. Grid is a multi-row list of database rows from a query. Graph displays a graph based on input values from the results set of a Query Object. A Graph can be defined to support a dates the user can change when the Graph is displayed. Another feature of reports, is the ability to drill down into a Transaction Object so that an item can be manipulated. Report displays a report based on
input values from the results set of a Query Object. A Report can be defined to support dates and other selectable criteria that the user can manipulate when the Report is displayed. Another feature of reports, is the ability to drill down into a Transaction Object so that an item can be manipulated. A Tree is a list of database rows, usually from several related queries that build the parent/child relationships. A Tree can coordinate with other views when nodes are clicked. The field, ‘CardHeadingField = [fieldname]’, specifies the name of the field that is used in the Card Heading if the SubObject1 parameter is Card. The field, ‘Dates = [FROM,TO]’, specifies that a Report or Graph should allow the user to customize dates based on a Start Date (FROM) and End Date (TO). Symbolic substitution via the [FROM] and [TO] parameters is supported in queries to dynamically change dates. The field, ‘DateField = [fieldname]’, specifies the name of the field that is used as the basis for the date range in a Report. If the Dates parameter is populated and DateField is not, a date range clause will not be appended to the ‘where’ clause of the primary report query. If both the ‘Dates’ parameter and the ‘DateField’ parameter are specified, then a date range is applied to the ‘where’ clause, to allow the user to customize dates based on a Start Date (FROM) and End Date (TO).

Continuing with reference to Figure 21a, the field ‘DDAllChoice = [YES | NO]’ describes whether a drop down list is used. All DDLxxx parameters refer to a DDList type field. A DDList is a Drop Down List that is populated by a query and will allow a user to limit the data displayed in any window to the rows that match the value of the DDList chosen by the user. DDAllChoice specifies whether an ‘All’ choice should be added to the top of the Drop Down List so the user could see ‘All’ rows returned. The field, ‘DDListField = [fieldname]’, is the name of the field in the results set to apply the drop down list filter to. The field, ‘DDListLabel = [labelname]’, specifies the name of the label that will appear next to the Drop Down List on the window. The field, ‘DDListQuery = [queryname]’, is the name of the Query Object to execute to populate this Drop Down List. The field, ‘DDNewChoice = [YES | NO]’, specifies if an ‘New’ choice should be added to the top of the Drop Down List so the user could add new selections to the list. The field, ‘DisplayNodeNames = [YES | NO]’, is used if the View is a tree to determine whether or not node names are displayed.

Continuing with reference to Figure 21a, the field ‘DefaultViewFormat = [fmt1, fmt2,..fmtn]’, provides a comma delimited list of view formats to apply. These values are defaults that the user can override using the customize button, or the calendar type buttons on a Calendar. The choices depend on the view type, for example: Grid, Zoom 80%,
Zoom 100%, Zoom 125% all allow the size of the grid to change. ‘No Vertical’, ‘No Horizontal’ indicate that horizontal and/or vertical lines will not be displayed. ‘Alternating Colors’, ‘White Background’, ‘Colored Background’ all relate to how the window will look in a grid. Calendar: Day, 5 Days, Week, Month all relate to the type of calendar displayed and the duration of time the calendar will show. The field, ‘DefaultTimeIncrement = [5 MINUTES...]’, specifies a time unit increment such as a single day or five day calendar view.

Continuing with reference to Figure 21a, the field ‘DetailView = [viewname]’, specifies the name of the View Object to display when a user drags an item from a DragDropList to this view. The values from the DragDropList are populated into the DetailView View where the user can manipulate them prior to closing the DetailView window and saving the values to this view. The field, ‘DrillDownListField = [fieldname]’, is one of the options that enables support for ‘Drill Downs’. Drilling down opens a Transaction Object and positions the user on the appropriate item, such as a row, in the view. Drilling down is activated by double clicking on a report item, clicking on a Tree Node, or using Action buttons and Action Menus. The fieldname specified is passed to the DDLList on the View to Drill Down into. The field, ‘DrillDownField = [fieldname]’, is similar to Drilling Down. Here, the fieldname that filters the View to drill down into is specified. The field, ‘DrillDownTab = [tablename]’, specifies the tablename that is selected on the Transaction to drilling down into. The field, ‘DrillDownTransaction = [transactionname]’, specifies the transaction name that is drilled down to. The field, ‘EqualColSize = [n]’, specifies a size that is used to equally space grid columns. The field, ‘Fields = [fieldname1,fieldname2…n]’, is used on Total Rows, such as in a Grid or on a MultiSection Report to specify a list of fields to operate on. The field, ‘FindFields = [fieldname]’, is used on a DragDropList or MultiColumn Custom Combo Box. A ‘Find All’ option is displayed, which will limit what is being seen to the characters type by the user. Specify the name of the field to filter by. The field, ‘Format = [GRID] DETAIL | CARD | LABEL’], is used on a MultiSection Report Section and specifies the format to apply to the report section. The Grid format is a table-like report that is the default format. The Detail format is like the detail view with a separate field name, field values on each line and blank values are displayed.

Continuing with reference to Figure 21a, a Card view is similar to detail view with a separate field name. The Label format will format standard laser printer labels. The
field, ‘GroupBy1 = [fieldname]’, is used on reports to specify which primary field to group
the report by. The value of the fieldname will only be displayed on the report at the
beginning of a new group. The query needs to be ordered by the groupbyfieldname1 so the
grouping is correct. The field, ‘GroupByAction1 = [SUM]’, is used on reports to specify to
SUM a group of records up until a new group occurs or the report. The field,
‘GroupByActionField1 = [fieldname]’, is used on reports to specify which field to apply the
Action specified in GroupByAction1 to. The field, ‘GroupBy2 = [fieldname]’, is used on
reports to specify which secondary field to group the report on. The value of the fieldname
will only be displayed on the report at the beginning of a new group. The field,
‘GroupByAction2 = [SUM]’, is used on reports to specify to SUM a group of records up until
a new group occurs or the report ends. The field, ‘GroupByActionField2 = [fieldname]’, is
used on reports to specify which field to apply the Action specified in GroupByAction1. The
field, ‘Instructions = [text]’, specifies the text to display on the top of a DragDropList. The
field, ‘MaxColSize = [n]’, specifies the maximum size to allow in a Grid Column when a
grid is first displayed. The field, ‘MinColSize = [n]’, specifies the minimum size to allow in
a Grid Column when a grid is first displayed. The field, ‘MultiSection = [YES | NO]’,
applies to Reports and indicates whether the report is MultiSectional. A MultiSectional
Report allows for multiple formats within a report, such as grids, card, details or other views.
Also, different queries can be executed for each section. The field, ‘MustFilter = [YES | 
NO]’, indicates that a Filter must be entered before displaying a View. The Filter is supplied
as part of the Customize button. A filter will limit the rows displayed in a view to rows that
match the filter criteria. The field, ‘NodeKey = [nodekeyfieldname]’, specifies the field name
of the node to use as a key to the node on a Tree View. The field, ‘NodeValue =
[nodevaluefieldname]’, specifies the field name from which to get the value to display on a
node of the Tree. The field, ‘OnInit = [FILTER]’, specifies that special processing should
take place before a view is displayed. The field, ‘Parent = [ROOT | NODENAME]’, specifies
the name of a parent node on a Tree View. The field, ‘ParentKey = [parentkeyfieldname]’,
specifies the field name of the key of the parent on a Tree View. The field, ‘Query =
[queryname]’, specifies the name of the Query Object to execute when the view displayed.
The field, ‘QueryStartDate = [fieldname]’, specifies the name of the field to use from
the Query to determine what is displayed on a Calendar View. The field,
‘QueryEndDate = [fieldname]’, specifies the name of the field to use from the Query to
determine what is displayed on a Calendar View. This field is used to show the duration of
time on a calendar. The field, ‘Title = [text]’, specifies the text to display as the Title on a
View. A user may specify a ‘controlledbyfieldname’ anywhere in the title to perform a
symbolic substitution of a field with the value that is coordinating the view.

Figure 21b shows a View Object 2194 that describes a grid GUI object
controlled by another GUI object. The Grid has a ‘TotalRow’ field that shows running totals.
Whenever ‘TotalRow’ changes, a special form of the Query ‘SetAllTotalFields’ will execute
and set all fields in the query to the values on the TotalRow.

With reference to Figure 22a, field level characteristics are defined with
respect to a view object 2202. The field, ‘Object = [VIEW]’, identifies the beginning of a
View Object definition. The field, ‘AppName = [appname]’, specifies the application name.
The field, ‘Name = [viewname]’, specifies the name of the View Object. This object is
referred to by the ViewName parameter on the Transaction Object. The field, ‘SubObject1 =
[FIELD]’, overrides the default field characteristics for this view. The field, ‘BlankWhenZero
= [Yes | No]’, is set to ‘Yes’ to display zeros as blanks or ‘No’ to show zeros. The field,
‘Calculate = [expression]’, defines the expression to execute when any FieldName specified
in the expression has changed. Supported operations include addition, subtraction,
multiplication and division. Each FieldName specifies a field from the current record to
include in the expression; Prev.FieldName indicates that a field from the prior row, such as in
a grid, should be used in the calculation; First.FieldName indicates that a field from the first
row, such as in a grid, should be used in the calculation; Last.FieldName indicates that a field
from the last row, such as in a grid, should be used in the calculation; ‘Now()’ indicates that
the current Date & Time should be used in the calculation.

The field, ‘ColorCondition = [< | > | =]’, specifies to change the color of the
text in the field when the condition is met. Conditions allowed are: ‘greater than’, ‘less than’,
equal to’, or ‘not equal to’. The field, ‘ColorConditionColor = [red | blue...]’, specifies the
color to be displayed when the ColorCondition is met. The color should be a value already
defined in the Color Object.

The field, ‘Control = [COMBOBOX | DATE...]’, specifies the type of control
to drop down when working in a field. Possible values include: CALCULATOR,
COMBOBOX, CHECKBOX, DATE, DATE/TIME, MAILTO, URL. The field,
‘ControlName = [controlname]’, specifies the name of the CustomControl Object. This is used to specify a specific custom control to use with the control.

The field, ‘DefaultValue = [Number | Now(). . .]’, specifies a default value to place in the field when a field is first initialized. The following parameter interpretations are used: ‘Number’ specifies a numeric value; ‘FieldName’ specifies a field from the current record to include in the expression; ‘Prev. Fieldname’ indicates that a field from the prior row in a grid should be used in the calculation; ‘First. Fieldname’ indicates that a field from the first row in a grid should be used in the calculation; ‘Last. Fieldname’ indicates that a field from the last row in a grid should be used in the calculation; ‘Now()’ indicates that the current Date & Time should be used in the calculation. The field, ‘DefaultColor = [red | blue. . .]’, specifies the color to use when the field is displayed. The default color should reference a value previously defined in a Color Object.

The field, ‘Format = [FLOAT | COMMA. . .]’, specifies the numeric format to apply to the field. Here, the following parameters are used: ‘FLOAT’ indicates a floating point number; ‘DOLLARS’ indicates a currency field; ‘COMMA’ specifies a numeric field with commas; and, ‘PERCENT’ specifies a numeric percent field. The field, ‘Hide = [YES | NO]’, hides the field from view, which is particularly useful for keys. The field, ‘Justification = [LEFT | RIGHT | CENTER]’, specifies where to position text within a cell. The field, ‘Places = [number]’, specifies the number of decimal places to display for a numeric field. The field, ‘Size = [number]’, specifies a field size in characters. The field, ‘Type = [LONG | SHORT. . .]’, specifies the data type of the field. Valid values include, but are not limited to, DOUBLE, SHORT, LONG, INT, DATE, DATETIME, TIMESTAMP, which describe data types that are known to one who is skilled in the art, and which may be extended to include further data types.

Figure 22b describes a field view object 2242 that overrides the default behavior for the field ‘Type’ and specifies that for the view ‘Customers Grid’ to default the field value to ‘0002’ when new rows are added.

Figure 23 shows a flow chart that provides a general overview of the process of creating an application program from the application program description objects.

Initially, in step 2302, the application shell is started and the application specification file is parsed to create an internal table of objects and their associated parameters. Here, the application specification file has been previously populated with a plurality of application
object descriptions. In the preferred embodiment the application specification file is implemented as two or more files, each containing a set of complementary application description objects. For example, all of the action objects, such as import, export query and execute command, may be stored in one file, while all of the descriptive objects such as GUI objects and transaction objects may be stored in a separate file. Thus, objects that invoke actions, which may be called verb objects, are distinguished from the descriptive objects, which may be called noun objects.

In step 2304, the Application Object is processed to determine global characteristics of the application program. A splash screen and user login screen are displayed, if specified in the application object. A data source, as specified in the application object, is also opened at this step. The data source may be an ODBC compliant database or other form of database system. The main application window, with non-dynamic components like the ToolBar, Statusbar and menu, is also created at this step.

In step 2306, the ‘Go’ menu is created to provide fast access to each of the transactions or views. The ‘Go’ menu is built as follows. First, FolderTab Objects are located in the internal object table. Then high-level menu groups are built for each item specified in the FolderTab Object using the sequence order specified therein. The Folder Objects are next located in the internal table. Finally, child menus are built for each high-level group specified in the FoldersTab Object by using the Folders Object with the specified sequence order.

In step 2308, the Folder View is processed as follows. Initially a Folder View window is created. Then Folder View Objects are located in the internal object table. A Folder Tree is built using the Parents specified in the Folder Object to create parent entries with the sequence order specified on the Folder Object. Child Folder entries are then created under each parent, using their specified sequence order.

In step 2310, a Home Page Object is created, using the Folder Objects in the internal objects table. The Home Page view is built using the specified parents to create parent entries in the sequence order specified by the Folder Object. Next, child views are created under each home page section in sequence order specified on the Folder Object.

In step 2312, the Outbar or shortcut menu is created as follows. First, an Outbar window is created. Next, the OutBar Objects are located in the internal objects table. Then the Outbar Groups are built in the sequence order specified by the Outbar Object. Then
icons are placed under each group specified on the Outbar Object, in sequence order. The icons are retrieved the specified Transaction Object.

In step 2314, previously stored user settings, if any exist, are parsed and previously opened Transaction Objects are reopened. As explained in more detail below, a third type of specification file is created and stored in some embodiments for storing specific settings and parameters set by a given user while using an instance of the generated application. This user settings file, if available, is parsed and each Transaction Object, which was open last time the application was run, is reopened. This provides application view persistence, so that open views remain consistent between sessions when the generated application is executed.

With reference to Figure 24, a flow chart is presented that describes the processing steps for displaying a view that was requested by a user. In step 2402, the user specifies, selects or activates a desired view by initiated a function in the application program. Once the Application Window is displayed a user can initiate a function by choosing from the OutBar, Folder, “Go” Menu or the Home Page views. In step 2404, transaction components are created for the selected transaction, including GUI object instances that may contain views of grid objects, card objects, tree objects or other fundamental object types. The GUI object instances, also called Transaction Components or Views, are created for the selected Transaction. The Transaction Objects are located in the internal objects table. The Transaction Objects define the number, and screen location, of windows to display and each Transaction Object specifies the one or more GUI Objects to be displayed.

In step 2406, complete views are assembled out of GUI objects as specified by the active Transaction Object. The GUI Objects or views are created based on each Transaction Object. The GUI Objects are located in the internal object table. Each view is created based on the View Object in the screen location specified in the Transaction Object.

In step 2408, color schemes and other attributes for the view are created. The background object is set to the Color Objects RGB color specified by the Transaction Object. Alternatively, the background is set to the Color specified on each BackGround Object for the Transactions’ FolderTab Object parent.

In step 2410, the views specified in each referenced GUI Object are created. Once the individual views specified by GUI objects are created, then the view is filled with data based on the Query Object that was stored with each GUI Object.
The GUI Object is then built and populated with data, step 2412. The Table Object is located in the internal object table for the Query, as specified on the associated GUI Object. The Table formatting and key information about the primary table is then obtained. The Query Object is located in the internal object table as specified by the associated GUI Object. The “Where” clause of the Query text is adjusted, if the View Object is being controlled by a user’s selection in a different screen section. The controlling requirements are specified by the Transaction Object. A SQL “Order By” clause is applied for statements that don’t have such a clause specified. The Query is then sent to the database for processing. The Field Object is then located for each field in the Query, using the field data type and formatting requirement specified on the Field Object or using the field Characteristics from the database driver to build the appropriate fields type on the View. The Field in the View is then set to the Control Type specified on the Field Object. Any ComboBoxes are filled with the Query Object specified on the Field Object. Any DDList Boxes are filled by the DDListQuery specified by the associated GUI Object. The Results Set from the database driver is used to fill the fields of the view with Data. The Last View Object, DDList value and field sizes are set to the values last used by this Transaction Object from the user setting file.

With reference to Figure 25, a flowchart describes view creation in more detail. In step 2502, the data source is located from the Application Object and the Query Object associated with this view. In step 2504, formatting and lookup table information is obtained from the application specification file. In step 2506, the Query Object associated with this GUI object is located. The text query is adjusted, in step 2508, so that it corresponds to the data source, and, in step 2510, the query is executed against the data source. In step 2512, the Field Object for the view is located. The field formatting is set to the value described in the field Object in step 2514. In step 2516, any associated combo boxes are filled. In step 2518, associated DDL lists are filled to allow fast access to the data displayed in this view. In step 2520 the field value itself is set so that the data may be displayed. In step 2522, the current view parameters are saved to a file so that the view may be restored. In step 2524, the view is displayed for the user to continue further interaction.

With reference to Figure 26, a flowchart describes the termination sequence of the application program. In step 2602, a list of open transaction objects is saved to a new user setting file. In step 2604, a transaction object for an open transaction is located. In step 2606,
the most recent query object, associated with the current transaction, is located. In step 2608, the last view of this transaction object is stored, including the type and status of associated GUI objects. In step 2610, the drop down lists associated with this transaction are stored. In step 2612, grid field sizes are stored, since they may have been modified by a user. In step 2614, the program tests to determine if there are any remaining transaction objects. If additional transaction objects remain, then the program control returns to step 2604 and the next transaction object is processed. Otherwise control progresses to step 2616 where the data source is closed. The program then terminates in step 2618.

With reference to Figure 27, an example of the operation of the data encapsulation system in a client/server environment is described. A server 2702 provides generated specification files 2706 and perhaps data source files 2704 in response to requests 2708 and 2712 from client stations 2714 and 2716, respectively. The requests signal whether to transmit a specification file 2706 alone, or a specification file 2706 with a data source file 2704. Here, the specification files 2706 for each request may be identical. For client 2714 the specification file 2706 is received and is used by the application generator 2720 along with locally stored application objects 2718 to create an application program 2730. The application program 2730 then queries the transmitted data source 2704 to populate object views with data.

For client 2716, only an application specification file 2706 is transmitted in response to request 2712. The application specification file 2706 is received by the client 2716 and interpreted by the application generator 2720. The application generator 2724 then uses the set of application level objects 2718, stored locally, and a locally stored data source 2704 to create the final application program 2730. If the transmitted data source 2704 and the local data source 2704, then the two generated application programs will also be identical, since, the specification files are identical. Thus, the application specification file is used to create a final application program independent of computing platform, data source location, or data source contents.

Figures 28-32 show screen displays from a generated application program 106 including component GUI objects 320. Figure 28 shows a screen display with a grid GUI object instance 2802 and an associated detail record 2803 showing further information for a selected grid row 2804. Here the application generation rules automatically link the grid view GUI object instance 2802 with the detail view GUI object instance 2804 allowing the
user to ‘drill-down’ into the grid GUI object instance. Figure 28 also shows a full application program, implemented as a stand alone browser, that includes a fully functioning menu bar 2806, short cut list 2808, window controls 2810, scroll bars 2812, resize controls 2814, a status line 2816 and other application program controls which may be determined at runtime by the specific target environment.

Figure 29 shows a generated application program screen with a graph GUI object 2930 displayed using the full application screen. Figure 30 shows a generated application program screen with a tree GUI object 3040 and homepage GUI object 3042 displayed. Figure 31 shows a generated application program screen with a calendar GUI object 3150, detail GUI object 3152, drag & drop GUI object 3154 displayed. Figure 32 shows a generated application program screen with a card GUI object 3262, detail GUI object 3264 and tree GUI object 3260 displayed.

While the invention has been described and illustrated in connection with preferred embodiments, many variations and modifications as will be evident to those skilled in this art may be made without departing from the spirit and scope of the invention, and the invention is thus not to be limited to the precise details of methodology or construction set forth above as such variations and modification are intended to be included within the scope of the invention.
WHAT IS CLAIMED IS:

1. A method for generating an application program for use in manipulating a first data source, the method comprising:
   storing a plurality of application level objects comprising:
   an application object for referencing a data source;
   a plurality of graphical user interface objects for defining a plurality of types of user interfaces by which data from a data source may be presented;
   a transaction object for referencing one or more of the graphical user interface objects and for defining a screen layout; and
   a data source object for defining characteristics of data in a data source;
   parsing an application specification file to retrieve a plurality of object characteristics including the first data source referenced in an application object, one or more first graphical user interface objects and a first screen layout;
   generating a plurality of application specific objects by applying the characteristics retrieved from the application specification file to the application level objects; and
   generating the application program from the application specific objects, the application thereby allowing manipulation of the first data source through the one or more first graphical user interfaces presented in accordance with the first screen layout.

2. The method of claim 1, wherein the step of storing the application level objects further comprises storing one or more activation objects for defining a mechanism to activate a transaction, and wherein the step of generating the application program comprises generating one or more visual indicators based on the one or more activation objects retrieved from the application specification file.

3. The method of claim 2 wherein a transaction object may be selected from the plurality of transaction objects based on visual indicators of the activation objects.

4. The method of claim 2 wherein the activation object comprises one or more folder objects each comprising:
   an object specifier for the folder object so that the folder object may be referenced by other objects;
   an application name for associating the folder object with an application object;
a tab name for displaying a title of the folder object to the user;
a parent identifier for locating a GUI object that references the folder object;

and,

a transaction name for associating a transaction object with the folder object.

5. The method of claim 2 wherein the activation object comprises one or more
outbar objects each comprising:

an object identifier for identifying the outbar object;
an application name identifier for associating the outbar with an application
program;

a name field for identifying the outbar object; and,

a transaction name for associating a transaction with the outbar.

6. The method of claim 1, wherein the step of generating the GUI objects
comprises selecting one or more graphical user interface object types from the group
consisting of a calendar GUI object; a grid GUI object; a card GUI object; a tree GUI object;
a drop down list GUI object; a drag & drop GUI object; a report GUI object; a graph GUI
object; and, a detail GUI object.

7. The method of claim 1, wherein a GUI object is constructed from the group
consisting of:

an application name identifier for associating the GUI object with an
application;

a GUI object name for identifying the GUI object so that it may be referenced
by other objects;

a GUI object type specifier that determines the GUI object type, the GUI
object type selected from the group consisting of: a calendar view object type specifier; a grid
view object type specifier; a card view object type specifier; a tree view object type specifier;
a drag & drop view object type specifier; a drop down list view object type specifier; a report
view object type specifier; a graph view object type specifier; a table view object type
specifier; and, a detail view object type specifier;

a drill down field that specifies a second GUI object to link to; and,

one or more field objects specifying content data to display.
8. The method of claim 7 wherein the calendar GUI object includes data fields comprising: date information for identifying a day or range of days; and, time information for specifying a time period.

9. The method of claim 7 wherein the grid GUI object includes data fields comprising: a multi-row list of grid data elements; grid row size information; and, grid column size information.

10. The method of claim 7 wherein the card GUI object includes data fields comprising: a list of card data elements; and, a card heading field.

11. The method of claim 7 wherein the tree GUI object includes data fields comprising: a hierarchical list of data elements; and, a parent object identifier.

12. The method of claim 7 wherein the drop down list GUI object includes a drop down list label that specifies the label that will appear next to the displayed drop down list.

13. The method of claim 7 wherein the drag & drop list GUI object includes instructions specifying text to display next to the drag & drop list.

14. The method of claim 7 wherein the graph GUI object includes a graph format specifier that determines the display type of the graph GUI object.

15. The method of claim 7 wherein the report GUI object includes section description information describing how to divide the report into one or more sections.

16. The method of claim 7 wherein the detail GUI object includes a detail view name that specifies the name of a GUI object to display when a user drags an item from a drag & drop GUI object view and drops it into this view.

17. The method of claim 7, wherein the table GUI object includes data fields comprising:

   a database name identifier for identifying a data source;
   a query name for processing data query operations to fill the table object with data from the data source;
   a reminder field that specifies whether the table object supports reminder processing;
   a plurality of reminder fields that specify fields to display when a reminder event is activated;
a plurality of transaction identifiers, each transaction identifier associated with a reminder field;
a flag that determines the status of any drop down lists;
an associated tab name that identifies the table object.

18. The method of claim 7, wherein the query GUI object includes data fields comprising: a query name for identifying the query; and, an SQL query for generating and executing SQL queries.

19. The method of claim 1, wherein the application level objects include display property objects selected from the group consisting of background objects and color objects.

20. The method of claim 19, wherein the background object includes data fields comprising:
an object name identifying the background object;
an application name associated with the background object; and,
a color identifier for determining a color to display.

21. The method of claim 19, wherein the color object includes data fields comprising:
an object identifier signifying that this object is a color object;
an application name associating the color object with an application object;
a name specifying the name of the color defined in the color object; and,
an encoded color value specifying the color defined by the color object.

22. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform method steps for generating an application program, the method comprising:
storing a plurality of application level objects comprising:
an application object for referencing a data source;
a plurality of graphical user interface objects for defining a plurality of types of user interfaces by which data from a data source may be presented;
a transaction object for referencing one or more of the graphical user interface objects and for defining a screen layout; and
a data source object for defining characteristics of data in a data source;
parsing an application specification file to retrieve a plurality of object characteristics including the first data source referenced in an application object, one or more first graphical user interface objects and a first screen layout;

generating a plurality of application specific objects by applying the characteristics retrieved from the application specification file to the application level objects; and

generating the application program from the application specific objects, the application thereby allowing manipulation of the first data source through the one or more first graphical user interfaces presented in accordance with the first screen layout.

23. A memory for storing data for access by an application program being executed on a data processing system, comprising:

a data structure stored in the memory, the data structure including:

an application level object specifying a data source;

a plurality of GUI objects for describing views of the content data; and

a plurality of transactions specifying one or more GUI objects and linking the one or more GUI objects to the data source and to other GUI objects;

where the data structure is used to generate an application program.

24. The memory of claim 23, wherein the data structure includes one or more activation objects for defining a command to activate a transaction object.
Fig. 4

1. Create Application Level Objects and a Data Source (402)
2. Store Application Specification File (404)
3. Parse Application Specification File (406)
4. Generate Application Specific Objects (408)
5. Generate Application Program from Application Specific Objects (410)
Fig. 5
Fig 6a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [APP]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [name]</td>
<td>Y</td>
</tr>
<tr>
<td>DBName = [odbcname]</td>
<td>Y</td>
</tr>
<tr>
<td>CompactFrequency = [nnn]</td>
<td></td>
</tr>
<tr>
<td>DatabaseSignon = [username</td>
<td>databaseusername</td>
</tr>
<tr>
<td>DatabasePassword = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>DBFileName = [dbfilename]</td>
<td></td>
</tr>
<tr>
<td>NewPassword = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>Registration = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>RegistrationPrefix = [xxxx]</td>
<td></td>
</tr>
<tr>
<td>RegistrationMessage = [xxxx]</td>
<td></td>
</tr>
<tr>
<td>RegistrationURL = [xxxx]</td>
<td></td>
</tr>
<tr>
<td>SaveDatabasePasswordOption = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>SavePasswordOption = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>SplashFile = [splashfilename]</td>
<td></td>
</tr>
<tr>
<td>SplashText = [splashtext]</td>
<td></td>
</tr>
<tr>
<td>UserDropDown = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>Version = [version]</td>
<td></td>
</tr>
<tr>
<td>VersionDate = [versiondate]</td>
<td></td>
</tr>
</tbody>
</table>

Fig 6b

### Fig 7a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [BACKGROUND]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>SubObject1 = [report</td>
<td>calendar...]</td>
</tr>
<tr>
<td>Color = [red</td>
<td>blue...]</td>
</tr>
<tr>
<td>SubObject2 = [title</td>
<td>heading...]</td>
</tr>
<tr>
<td>Name = [foldertabobject]</td>
<td></td>
</tr>
</tbody>
</table>

### Fig 7b

- AppName=[MyApp]
- Object=[Background]
- SubObject1=[Calendar]
- SubObject2=[DayHeadingFocusDay]
- Color=[SkyBlue]

### Fig 8a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [COLOR]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [colorme]</td>
<td>Y</td>
</tr>
<tr>
<td>RGB = [r,g,b]</td>
<td>Y</td>
</tr>
</tbody>
</table>

### Fig 8b

- AppName=[All]
- By=[A]
- Object=[Color]
- Name=[Red]
- RGB=[255,0,0]
Fig 9a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [EXECUTECOMMAND]</td>
<td>Y</td>
</tr>
<tr>
<td>Command = [commandtext]</td>
<td>Y</td>
</tr>
</tbody>
</table>

Fig 9b

Object=[ExecuteCommand] Command=[COPY ABC.TXT EDF.TXT]

Fig 10a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [EXECUTEQUERY]</td>
<td>Y</td>
</tr>
<tr>
<td>Query = [queryname]</td>
<td></td>
</tr>
</tbody>
</table>

Fig 10b

Object=[ExecuteQuery] Query=[obj]

Fig 11a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [EXPORT]</td>
<td>Y</td>
</tr>
<tr>
<td>Query = [queryname]</td>
<td></td>
</tr>
<tr>
<td>FileName = [drivepathfilename]</td>
<td>Y</td>
</tr>
<tr>
<td>FileType = [CSV</td>
<td>ZGC]</td>
</tr>
</tbody>
</table>

Fig 11b

**Fig 12a**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [FIELD]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [fieldname]</td>
<td>Y</td>
</tr>
<tr>
<td>Query = [queryname]</td>
<td></td>
</tr>
<tr>
<td>BlankWhenZero = [Yes</td>
<td>No]</td>
</tr>
<tr>
<td>Calculate = [expression]</td>
<td></td>
</tr>
<tr>
<td>ColorCondition = [&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>ColorConditionColor = [red</td>
<td>blue…)</td>
</tr>
<tr>
<td>Control = [COMBOBOX</td>
<td>DATE…)</td>
</tr>
<tr>
<td>ControlName = [controlname]</td>
<td></td>
</tr>
<tr>
<td>DefaultValue = [Number</td>
<td>Now()…)]</td>
</tr>
<tr>
<td>DefaultColor = [red</td>
<td>blue…)</td>
</tr>
<tr>
<td>Format = [FLOAT</td>
<td>COMMA…)</td>
</tr>
<tr>
<td>Hide = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>Justification = [LEFT</td>
<td>RIGHT</td>
</tr>
<tr>
<td>Places = [number]</td>
<td></td>
</tr>
<tr>
<td>Size = [number]</td>
<td></td>
</tr>
<tr>
<td>Type = [LONG</td>
<td>SHORT…)</td>
</tr>
</tbody>
</table>

**Fig 12b**


**Fig 13a**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [FOLDERTAB]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [foldertabname]</td>
<td>Y</td>
</tr>
<tr>
<td>Sequence = [number]</td>
<td></td>
</tr>
</tbody>
</table>

**Fig 13b**

Fig 14a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [FOLDER]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [foldername]</td>
<td>Y</td>
</tr>
<tr>
<td>TabName = [foldertabname]</td>
<td>Y</td>
</tr>
<tr>
<td>Sequence = [number]</td>
<td></td>
</tr>
<tr>
<td>Parent = [foldername]</td>
<td></td>
</tr>
<tr>
<td>Transaction = [transactionname]</td>
<td></td>
</tr>
</tbody>
</table>

Fig 14b


Fig 15a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [IMPORT]</td>
<td>Y</td>
</tr>
<tr>
<td>TableName = [table]</td>
<td>Y</td>
</tr>
<tr>
<td>FileName = [drivepathfilename]</td>
<td>Y</td>
</tr>
<tr>
<td>FileType = [CSV</td>
<td>ZGC]</td>
</tr>
</tbody>
</table>

Fig 15b

Object=[Import] TableName=[Customers] FileName=[customers.csv] FileType=[CSV]
Fig 16a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>[OUTBAR]</td>
</tr>
<tr>
<td>AppName</td>
<td>[appname]</td>
</tr>
<tr>
<td>Name</td>
<td>[outbar]</td>
</tr>
<tr>
<td>TabName</td>
<td>[follertablename]</td>
</tr>
<tr>
<td>Sequence</td>
<td>[number]</td>
</tr>
<tr>
<td>Parent</td>
<td>[foldername]</td>
</tr>
<tr>
<td>Transaction</td>
<td>[transactionname]</td>
</tr>
</tbody>
</table>

Fig 16b


Fig 17a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>[QUERY]</td>
</tr>
<tr>
<td>AppName</td>
<td>[appname]</td>
</tr>
<tr>
<td>Name</td>
<td>[queryname]</td>
</tr>
<tr>
<td>DetailTable</td>
<td>[tablename]</td>
</tr>
<tr>
<td>SQL</td>
<td>[sqlquery]</td>
</tr>
</tbody>
</table>

Fig 17b

AppName=[MyApp]  By=[M]  Object=[Query]  Name=[Delete Items]  DetailTable=[Items]  SQL=[Delete from [Items] where Item = '%F[Item]']
### Fig 18a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [STARTUP]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>SubObject1 = [folder</td>
<td>outbar...]</td>
</tr>
<tr>
<td>SubObject2 = [default]</td>
<td>Y</td>
</tr>
<tr>
<td>Sequence = [number]</td>
<td></td>
</tr>
<tr>
<td>Hide = [YES</td>
<td>NO]</td>
</tr>
</tbody>
</table>

### Fig 18b


### Fig 19a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [TABLE]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [fieldname]</td>
<td>Y</td>
</tr>
<tr>
<td>SubObject1 = [define</td>
<td>Action]</td>
</tr>
<tr>
<td>SubObject2 = [afterinsert</td>
<td>afterdelete...]</td>
</tr>
<tr>
<td>Sequence = [number]</td>
<td></td>
</tr>
<tr>
<td>Query = [queryname]</td>
<td></td>
</tr>
<tr>
<td>PrimaryKey = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>PrimaryKeyHidden = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>Recurring = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>Reminder = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>ReminderFields = [fieldname1,fieldname2...]</td>
<td></td>
</tr>
<tr>
<td>ReminderTran = [transactionname]</td>
<td></td>
</tr>
<tr>
<td>ReminderTranDDLValue = [ALL</td>
<td>ddlvalue]</td>
</tr>
<tr>
<td>ReminderTranTabName = [tablename]</td>
<td></td>
</tr>
</tbody>
</table>

### Fig 19b

### Fig 20a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [TRANSACTION]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [fieldname]</td>
<td>Y</td>
</tr>
<tr>
<td>SubObject1 = [define</td>
<td>TopView1...</td>
</tr>
<tr>
<td>Sequence = [number]</td>
<td></td>
</tr>
<tr>
<td>TabName = [tabname]</td>
<td></td>
</tr>
<tr>
<td>Color = [red</td>
<td>blue...]</td>
</tr>
<tr>
<td>FarRightPercent = [n]</td>
<td></td>
</tr>
<tr>
<td>IconName = [iconname]</td>
<td></td>
</tr>
<tr>
<td>IndependentPercent = [n]</td>
<td></td>
</tr>
<tr>
<td>LeftPercent = [n]</td>
<td></td>
</tr>
<tr>
<td>Orientation = [HORIZONTAL</td>
<td>VERTICAL]</td>
</tr>
<tr>
<td>TopPercent = [n]</td>
<td></td>
</tr>
<tr>
<td>ViewName = [viewname]</td>
<td></td>
</tr>
<tr>
<td>ControlledByField1 = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>ControlledByField2 = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>ControllingField1=[fieldname]</td>
<td></td>
</tr>
<tr>
<td>ControllingField2=[fieldname]</td>
<td></td>
</tr>
<tr>
<td>ControlsView1=[viewname]</td>
<td></td>
</tr>
<tr>
<td>ControlsView2=[viewname]</td>
<td></td>
</tr>
<tr>
<td>ControlsView2=[viewname]</td>
<td></td>
</tr>
<tr>
<td>TabStyle = [T</td>
<td>D]</td>
</tr>
</tbody>
</table>

### Fig 20b

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [VIEW]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [fieldname]</td>
<td>Y</td>
</tr>
<tr>
<td>SubObject1 = [DEFINE</td>
<td>DATES</td>
</tr>
<tr>
<td>SubObject2 = [default</td>
<td>report...]</td>
</tr>
<tr>
<td>CardHeadingField = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>Dates = [FROMTO]</td>
<td></td>
</tr>
<tr>
<td>DateField = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>DDAllChoice = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>DDLListField = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>DDLListLabel = [labelname]</td>
<td></td>
</tr>
<tr>
<td>DDLListQuery = [queryname]</td>
<td></td>
</tr>
<tr>
<td>DDNewChoice = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>DisplayNodeNames = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>DefaultViewFormat = [fmt1,fmt2,...fmtn]</td>
<td></td>
</tr>
<tr>
<td>DefaultTimeIncrement = [5 MINUTES...]</td>
<td></td>
</tr>
<tr>
<td>DetailView = [viewname]</td>
<td></td>
</tr>
<tr>
<td>DrillDownDDLListField = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>DrillDownField = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>DrillDownTab = [tabname]</td>
<td></td>
</tr>
<tr>
<td>DrillDownTransaction = [transactionname]</td>
<td></td>
</tr>
<tr>
<td>EqualColSize = [n]</td>
<td></td>
</tr>
<tr>
<td>Fields = [fieldname1,fieldname2...n]</td>
<td></td>
</tr>
<tr>
<td>FindFields = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>Format = [GRID</td>
<td>DETAIL</td>
</tr>
<tr>
<td>GroupBy1 = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>GroupByAction1 = [SUM]</td>
<td></td>
</tr>
<tr>
<td>GroupByActionField1 = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>GroupBy2 = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>GroupByAction2 = [SUM]</td>
<td></td>
</tr>
<tr>
<td>GroupByActionField2 = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>Instructions = [text]</td>
<td></td>
</tr>
<tr>
<td>MaxColSize = [n]</td>
<td></td>
</tr>
<tr>
<td>MinColSize = [n]</td>
<td></td>
</tr>
<tr>
<td>MultiSection = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>MustFilter = [YES</td>
<td>NO]</td>
</tr>
<tr>
<td>NodeKey = [nodekey:fieldname]</td>
<td></td>
</tr>
<tr>
<td>NodeValue = [nodevalue:fieldname]</td>
<td></td>
</tr>
<tr>
<td>OnInit = [FILTER]</td>
<td></td>
</tr>
<tr>
<td>Parent = [ROOT</td>
<td>NODENAME]</td>
</tr>
<tr>
<td>ParentKey = [parentkey:fieldname]</td>
<td></td>
</tr>
<tr>
<td>Query = [queryname]</td>
<td></td>
</tr>
<tr>
<td>QueryStartDateTime = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>QueryEndDateTime = [fieldname]</td>
<td></td>
</tr>
<tr>
<td>Title = [text]</td>
<td></td>
</tr>
</tbody>
</table>
Fig 21b

Name=[Orders Grid]  DetailView=[Orders Detail]  Query=[Orders]  Title=[Details for &Order&]

Fields=[Price, Quantity]  Query=[Update Foods SetAllTotalFields where Food = '%F[Order]' ]
Title=[Order Totals]

Fig 22a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object = [VIEW]</td>
<td>Y</td>
</tr>
<tr>
<td>AppName = [appname]</td>
<td>Y</td>
</tr>
<tr>
<td>Name = [viewname]</td>
<td>Y</td>
</tr>
<tr>
<td>SubObject1 = [FIELD]</td>
<td>Y</td>
</tr>
<tr>
<td>SubObject2 = [DEFAULT]</td>
<td>Y</td>
</tr>
</tbody>
</table>
| BlankWhenZero = [Yes | No]
| Calculate = [expression] |
| ColorCondition = [< | > | =] |
| ColorConditionColor = [red | blue...]
| Control = [COMBOBOX | DATE...]
| ControlName = [controlname] |
| DefaultValue = [Number | Now()...]
| DefaultColor = [red | blue...]
| Format = [FLOT | COMMA...]
| Hide = [YES | NO]
| Justification = [LEFT | RIGHT | CENTER]
| Places = [number]     |
| Size = [number]       |
| Type = [LONG | SHORT...]

Fig 22b

SubObject2=[Default]  Name=[Customers Grid]  SubObject1Name=[Type]
DefaultValue=[0002]
Fig. 23

1. Parse the Application Specification File and Create a Table of Stored Objects
2. Process Application Object
3. Create the 'Go' Menu
4. Create the Folder View
5. Create the Home Page View
6. Create the Outbar View
7. Parse User Settings and Reopen Previously Opened Transaction Objects
Fig. 24

1. User Initiates a Function
2. Create Transaction Components
3. Create Views Based on Transactions
4. Create Color Schemes and View Attributes
5. Build View Specified in GUI Object
6. Fill the View with Data
Fig. 25

1. Locate Data Source
2. Obtain Formatting and Lookup Information
3. Locate Query Object Associated with View
4. Adjust Query Text Location
5. Query the Data Source
6. Locate Field Object
7. Set Field Formatting In View
8. Fill Any Associated Combo Boxes
9. Fill Any Associated DDL Boxes
10. Set Field Value
11. Set Last View Object, DDL Value, and Field Sizes to the Last Used Values Stored in Preferences File
12. Generate Application Program from Application Specific Objects
Fig. 26

1. Save List of Open Transaction Objects to User Settings File
2. Get Next Transaction Object
3. Locate Query Object
4. Save Last View Object for this Transaction
5. Save DDL Selections
6. Save Grid Field Sizes
7. More Transactions?
8. Close Data Source
9. Exit
Fig. 30
INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER
  IPC(7) : G06F 3/00
  US CL : 345/339, 356, 968
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
  U.S. : 345/339, 352,353,356,356,968; 717/1,2; 707/3,4,5

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
BRS, PROQUEST, ACM

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, P</td>
<td>US 6,075,530 A (LUCAS et al) 13 June 2000 (13.06.2000), col. 3, lines 14-17 and lines 25-63; col. 5, lines 38-52; col. 6, lines 11-23; col. 7, lines 27-66; col. 12, lines 9-22; claim 2; col. 8, line 47 - col. 9, line 33; col. 7, lines 6-9; col. 10, lines 11-14.</td>
<td>1-28</td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of Box C. See patent family annex.

Date of the actual completion of the international search

Date of mailing of the international search report

Name and mailing address of the ISA/US
  Commissioner of Patents and Trademarks
  Box PCT
  Washington, D.C. 20231
  Facsimile No. (703)305-3230

Authorized officer
  James R. Matthews
  Telephone No. (703) 305-9970

Form PCT/ISA/210 (second sheet) (July 1998)