



SynApp2.org

SynApp2 Concepts

Web Application Generator Reference

| | |
|---|----|
| SYNAPP2 CONCEPTS | 3 |
| <i>Introduction</i> | 3 |
| <i>Overview</i> | 3 |
| <i>Model - View - Controller (MVC) Architecture</i> | 4 |
| Model..... | 4 |
| Relations..... | 6 |
| View | 9 |
| Interactive Page Flow | 10 |
| Navigation Paths..... | 10 |
| Forms..... | 12 |
| Form Roles..... | 14 |
| Screen Image..... | 17 |
| Display Size and Page Layout..... | 17 |
| About Browsers | 20 |
| Controller | 21 |
| Exchange Cycle | 21 |
| Events | 23 |
| Page Load | 23 |
| Page Navigation and Report Tabs..... | 24 |
| Search..... | 26 |
| Reset..... | 27 |
| Browse (Row/Record Navigation) | 28 |
| Select | 29 |
| Add | 32 |
| Edit | 39 |
| Delete | 40 |
| Close..... | 42 |
| Statistics | 42 |
| <i>Templates</i> | 44 |
| <i>Reports</i> | 44 |
| <i>Custom Processes</i> | 46 |
| <i>Debug Message Window</i> | 47 |
| <i>Event Message Window</i> | 48 |
| <i>Hard Error Reporting</i> | 48 |
| <i>Installation and Deployment</i> | 49 |
| Configuration | 50 |
| Authentication and Authorization | 51 |
| Synapp2 Method | 51 |
| Direct Method | 53 |
| App and Enterprise Methods..... | 53 |

SynApp2 Concepts

The concepts presented below are intended for SynApp2 developers who need a deeper understanding of how SynApp2 works, in order to apply customization or modify the source code. Much of the terminology used in the descriptions appears within the SynApp2 source code and in pages generated by the software.

Introduction

The *SynApp2 - Web Application Generator* framework and the applications created with it, rely on popular, broadly supported technology. Every related component is available from both commercial providers and as [free and open source software](#).

Virtually all modern personal computers or workstations have web browser software installed. That is all that is needed to access and use applications developed with SynApp2.

The framework and applications created with it can be hosted on an individual workstation, a server on a local area network, wide area network, or the Internet.

The web-based implementation model centralizes data and reduces issues related to traditional application software distribution, licensing and maintenance costs, to virtually zero.

The interactive elements are implemented using strictly W3C standards-compliant Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS).

The programming logic is divided between bodies of code written in JavaScript and PHP. Persistent data is managed by popular database engines such as MySQL and Oracle. All of this, in turn, is supported by any of several web-server programs, including, but not limited to, Apache HTTP Server and Microsoft Internet Information Services (IIS).

All of the industry standard technical knowledge and skills needed to create, support and maintain SynApp2 generated applications is available from a huge pool of professionals. And, because SynApp2 is open source software, you, your organization or company, can augment the framework and applications if and as needed.

Overview

The SynApp2 approach to application development involves a series of small, simple solutions that target specific elements of a larger problem. When all, or most, of the solutions are applied, the result is an application program.

Breaking a larger problem into a series of smaller problems often has benefits. Allowing automation to deal with much of the work frees your time and lets you focus on key issues. The framework can provide support for more complex GUI or processing, and the means to integrate those pieces into a complete application.

SynApp2 divides problem areas and neatly exposes opportunities to tie into an orchestrated stream of events and responses. Some *hooks* are formally set out, while others are visible,

ready and waiting for a reason to be brought into use. Because SynApp2 is open source software, you can tie into whatever makes sense for your situation.

There are numerous areas of functionality that SynApp2 competently and efficiently addresses. Daily, untold numbers of developers with all levels of experience, spend countless hours of time and energy, trying to reinvent solutions for those same problems. Sadly, the results of much of that effort are nowhere near as nice as the facilities SynApp2 provides.

SynApp2 is so lightweight and unobtrusive that you can combine it with other tools and frameworks. As long as you've got PHP and a database (with numeric keys) accessible with one of the supported engines, you can add SynApp2 to the mix. Use SynApp2 to create complete applications, design prototypes, perform maintenance, or anything else you can think of, and to get done what you need to get done.

The purpose of SynApp2 is to help you create and deploy useful applications quickly, with the fewest number of details to deal with and remember, and that stand up over time with minimum support burden.

Print a hardcopy, get out your highlighter pen, and carefully sift through this document. It's loaded with useful information that will help you learn about and harness SynApp2.

Model - View - Controller (MVC) Architecture

The elements of the SynApp2 framework are segregated into the three distinct domains of *model*, *view* and *controller*. Discussions of this common architectural pattern for software engineering abound. Here, we'll focus on what the domains mean for SynApp2 generated applications and to the usefulness of the framework in other contexts.

Model

The **model** is primarily based on the data upon which the generated application operates. More precisely, model information is extracted from the database structure, as it is available from a *live instance* of the database, and made available by way of a program interface that defines useful operations in terms of meaningful conceptual abstractions.

Model information, beyond what is intrinsic to the database structure, may be augmented by supplemental expressions. This augmentation happens through a process called *customization*. The business logic of your application, beyond what is expressed by the database design itself, is implemented with a series of SQL and/or PHP code *snippets*. You'll see examples as you read on.

The model domain is itself implemented as several layered components and is subjected to influences from a number of sources, and produces specific outputs.

At the bottom of pile is the database extraction layer. This module, implemented in `_shared_/dbx.php`, abstracts fundamental database operations and encapsulates the

implementation details needed to work with different SQL database engines. At the time of this writing, MySQL and Oracle are supported.

[Aside: If anyone is willing and able to sponsor development for another engine, we're ready to go. Please contact richard@synapp2.org.]

Without going into too much detail, the database abstraction layer manages database connection details, and retrieves, generalizes and publishes database structure information, submits database queries (formulated elsewhere), and publishes generalized results in PHP variables or associative arrays.

For nearly all purposes, code within the framework or from custom processing functions uses a globally visible function – `get_dbx()` – to retrieve a reference to a reusable instance of a database abstraction object. See `_shared_/util.php`.

Practical examples of six of the most useful methods of the database abstraction object:

```
$customer_count = get_dbx()->get_query_value('select count(*) from customer');
```

```
$customer_recs = get_dbx()->get_query_array('select lastname, firstname from customer order by lastname, firstname');
```

```
function check_valid_date($d)
{
    $ymd = get_dbx()->parse_date_value($d); //MySQL [yyyy-mm-dd], Oracle [dd Mmm yy]

    return checkdate($ymd['m'], $ymd['d'], $ymd['y']);
}
```

```
$q_insert = get_dbx()->get_insert_id_query('id', "insert into schedule (identifier,
revision) values ('{$schedule_beg}', '{$revision}')");

get_dbx()->query($q_insert);

$id_schedule = get_dbx()->insert_id();
```

Notice the last three statements above. Such a sequence works on any of the supported database engines. The program code necessary to retrieve the primary key value of an inserted record is dramatically different for MySQL than it is for Oracle. The PHP code sequence using `dbx`, particularly involving Oracle, is simpler. It's portable too. If you ever have to switch database engines, or deploy on several different servers, with different database engines, you're covered with a single code base.

The next layer up in the model domain is the *schema* object. This module, implemented in `_shared_/schema.php`, provides an interface to the structure of the database in terms of database, table and column names, column types, and sizes. Also, a [graph](#) of the data table relations is maintained. Functions to navigate parent-child relationship hierarchy are available, and are used to drive the generation of GUI elements and SQL statements.

Relations

Any combination of three techniques may be used to declare how tables, or more specifically, how the records within those tables relate. At the center of each declaration technique is the notion and names of primary and foreign keys. We're not going to discuss relational database theory, *or philosophy*, here. There are other sources for that.

We need to touch on a couple of things before we go into declaring relations.

First, primary and foreign keys, as far as the SynApp2 MVC framework is concerned, are numeric. We'll acknowledge here and now, that for some, this is intolerable. But, for others, especially where new projects are concerned, strictly numeric keys are not only practical; they may prove to be positively liberating.

The column serving as the primary key should be specified in the SQL table definition.

Primary key values are established at the database level. MySQL often relies on its `AUTO_INCREMENT` feature, whereas Oracle will need a `sequence` and `trigger`. However you want to do it, when a record is inserted, a primary key value gets generated and assigned.

You'll need to do something like the following (once) for each table (Oracle only):

```
create sequence "customer_SEQ";

create trigger "BI_customer" before insert on "customer"
  for each row
  begin
    select "customer_SEQ".nextval into :NEW.id from dual;
  end;
```

Relations are established by declaring that the row value in a particular column, a so called foreign key (FK), corresponds to the primary key (PK) column value in a row of a parent table. How you do this depends on what fits your situation.

You may of course, and probably should, declare foreign key constraints according to the capabilities of your database engine. But, because of differences in how these details would be retrieved from a given engine, no attempt is currently made by the framework to do this – at least for now. Whether or not foreign key constraints are enforced by the database engine, The SynApp2 framework tests and enforces foreign key integrity as records are inserted or deleted with automatically generated query statements.

The easiest way to declare relations is with a column naming convention. SynApp2 recognizes the column name `'id'` to be the primary key column. The primary key is also detected, if one has been specified in the table definition, but this is not an absolute requirement.

A naming convention allows a foreign key to be declared with the combination of a prefix and the name of the table containing the record to which the foreign value refers. When the foreign key prefix `'id_'` is combined with a table name, a name like `'id_customer'`, is taken to mean that the column value is a foreign key into the customer table. If the `'id_customer'` column is in a table named `'orders'`, it implies a one-to-many relationship between customer

and orders, i.e., one *customer* record may be associated with many *orders* records. Some readers may find the term *master-detail* relationship to be more familiar.

The naming convention for primary key columns is: 'id'

The naming convention for foreign key columns is: 'id_<join_table_name>'

You can build relations into your application with entries in the `custom.inc.php` file:

```
$this->m_data[DATABASE]['sample'][KEYMAP][] =  
    array('TABLE_NAME'=>'orders',  
          'COL_NAME'=>'customerID',  
          'JOIN_TABLE'=>'customer',  
          'JOIN_COL'=>'customerPK');
```

You can also use the SynApp2 KeyMap page to interactively specify relations.

Select a column with a click on the appropriate radio button and then choose the join table (or delete) from the drop-down list. The display will immediately reflect the change.

SynApp2 KeyMap

SynApp2

KeyMap Options PageGen Reports Logout

AppID (or database): sample QueryID (or table): orders

sample.orders

| | Col Name | Join Table | Join Col |
|----------------------------------|------------------|------------|------------|
| <input checked="" type="radio"/> | orderID (PK) | orders | orderID |
| <input type="radio"/> | customerID (FK1) | customer | customerID |
| <input type="radio"/> | order_no | | |
| <input type="radio"/> | order_addr1 | | |
| <input type="radio"/> | order_addr2 | | |
| <input type="radio"/> | order_city | | |
| <input type="radio"/> | order_state | | |
| <input type="radio"/> | order_zip | | |
| <input type="radio"/> | order_date | | |
| <input type="radio"/> | ship_date | | |

customer.customerID
-- delete --

Use SynApp2 KeyMap for non-conforming key column names

A *system* table with the reserved name `_keymap_`, will be created automatically in the database you're working with.

The automatically generated SQL code needed to establish the mapping table varies, depending on database engine, but the following pseudo-code conveys the concept:

```
CREATE TABLE _keymap_ (
  id int(10) unsigned NOT NULL auto_increment,
  table_name varchar(32) NOT NULL,
  col_name varchar(32) NOT NULL,
  join_table varchar(32) NOT NULL,
  join_col varchar(32) NOT NULL,
  PRIMARY KEY (id)
);
```

You must have table create privileges in order for this method to work. For MySQL and Oracle the requirements are different. For MySQL, on a local server it is usually the *synapp2* user, or on a remote server the database *admin-user*, that must have the privileges. For Oracle, it is the (directly logged in/owner) user of the schema that needs table create privileges.

Regardless of the method you choose to define relations, once they're established, the SynApp2 framework uses them to build SQL statements.

An SQL statement with key columns that conform to SynApp2 naming conventions:

```
select orders.id,
       orders.id_customer,
       concat(customer.last_name, ' ', customer.first_name) as orders_id_customer,
       orders.order_no,
       orders.order_addr1,
       orders.order_addr2,
       orders.order_city,
       orders.order_state,
       orders.order_zip,
       orders.order_date,
       orders.ship_date,
       orders.id_order_mode,
       order_mode.descr as orders_id_order_mode,
from orders
left join customer on customer.id = orders.id_customer
left join order_mode on order_mode.id = orders.id_order_mode
where orders.id_customer='1019'
order by orders.id
);
```

Notice how the primary and foreign key column names appear in the `join` clauses, above.

The model-driven SynApp2 query generator is implemented in `_shared_/query.php`. Column expressions can be added with `MACRO`, `FETCH`, `EXTRA`, and `SUBQUERY` customization entries. See [SynApp2 Customization](#). Any join clauses, needed to support the resulting query are generated automatically. Read that last sentence again.

A two-stage data validation processor is implemented in `_shared_/validate.php`. And although data validation is substantially carried out by the model, an illustrated discussion appears later in this document. There's an extensive technical discussion, with examples, that can be found in the [SynApp2 Customization](#) document.

View

The **view** elements are implemented with HTML and CSS, particularly for SynApp2 generated pages. These elements are managed by some JavaScript code that interfaces with, but is not part of the controller domain – discussed later.

The term *view*, here, is taken to mean the GUI – the visible parts of the application – aside from the data being acted upon. The GUI elements are comprised of panels, or *forms*, that appear in different *roles* according to various states that occur in response to user interaction. There are also page and report navigation *tabs* that provide access to various parts of the application.

Other technology, such as Java, Flash, Silverlight, or even HTML 5, could be used on pages created by means other than SynApp2, but still be supported by the rest of the SynApp2 framework. Further discussion of that subject is beyond the scope of this document.

SynApp2 can, and typically does, generate all view elements of an application. Put another way, SynApp2 generates complete applications. Everything that is needed is produced at the click of a button. Without any information, other than a database instance, SynApp2 will automatically create an entire application, including PDF reporting and data export.

Details, such as which, or in what order, fields and columns appear on particular forms, and in various roles, are designated by *options* or by *customization*. These import topics are discussed later in this document. They're also addressed by the [SynApp2 Customization](#) document.

Markup templates are used for the page structure and style. But, as of this writing, certain aspects of the generated elements are imposed. Conformance with certain assumptions and conventions is needed, to make all the pieces work together. The code that emits all of the generated GUI elements can be found in `_shared_/markup.php`.

While the framework supports dynamic GUI generation, all of the GUI *forms* that SynApp2 generates are static HTML. This means that you can use readily available markup editing tools to stylize them. Do this as a final step, after the design of your database and application has been proven.

Almost all of what SynApp2 generates is view related – GUI. Very little executable code is emitted. SynApp2 generated executable code is limited to a few, simple, name vs. value mapping statements. It is the generalized, application-independent executable code of the underlying SynApp2 MVC framework that is reused, or *shared*, by every SynApp2 powered application. There will be more on this later, but for those who are leery of code generators, take comfort. Amazing things can be done with the SynApp2 MVC framework, beyond just supporting automatically generated applications.

It is also worth mentioning that, not only are the implementation details of the GUI, but even the notion of existence of a/the/any *view*, is completely independent and hidden from, the other domains of the framework.

Interactive Page Flow

In order for a web app to feel and act like a custom made, dedicated business application program, there really should be some kind of logical flow and guidance for users. Contrast this with the idea of a blank page in a word-processor or an empty spreadsheet.

SynApp2 processes details about your database structure and generates elements to support interactive management of your application data in a logical, sensible way. In order to do this, many aspects of relational database-backed applications have been abstracted and generalized.

SynApp2 depends on your overall database structure, or *schema*, to reflect the business problem you're trying to solve. SynApp2 strives to insure that the referential integrity of your database is maintained. This ambition shapes how your application can and will work.

The pages SynApp2 generates each have as their focus, a so-called *basis table*. The typical mission of a SynApp2 generated page is to support: *create*, *read*, *update* and *delete* (*CRUD*) operations on the records in the basis table. Another, but somewhat less well known acronym more completely describes the fundamental operations supported the *scaffolding* produced by the application generator: *browse*, *read*, *edit*, *add* and *delete* (*BREAD*).

Navigation Paths

On any given page, SynApp2 will typically impose a flow of user interaction that begins in the top left of the screen and flows toward the bottom, in a kind of column. On more complex pages, there can be more than one column of interaction flowing toward the bottom of the page. A SynApp2 page is laid out, more or less, as a grid.

These columns of interaction correspond to what SynApp2 calls paths. A path is an abstraction, depicting a chain of parent-child relationships between [the records in] tables. The number of paths *to* a table is equal to the number of foreign keys [it has] defined in its *keymap*. A *navigation path* is a specialized derivation of the path concept as it applies to interactive page flow.

As has been said elsewhere, the focus of a SynApp2 page is its *basis table*. The number foreign keys defined for the basis table is a significant determinant of the number of navigation paths needed for a given page. But, another major factor is whether or not you designate a *tier table* for a second or subsequent path (foreign key) with PageGen.

If there is only one path (foreign key) for the basis table, the number of paths, and therefore the number of navigation paths, will always be one.

When there are two or more navigation paths, the supporting forms will be situated in columns flowing across the page to the right.

Paths, and therefore navigation paths, also have a depth. The depth of a path is fixed by the chain of parent-child relationships. A navigation path can be, and often is, shorter than the full hierarchy of relationships defined by the schema, depending upon designation of a tier table.

The points in the interactive page flow that affect a given table along a particular path, or within a column, can be thought of as *nodes* of interaction.

Typically, the first node of interaction on a page occurs with a *Search Form* at the top of the leftmost column of forms and is on the *primary* path. The active node of interaction generally moves down the page, until the basis table is reached, and you are manipulating records in the basis table from events initiated by controls on the related *Select Form*.

Whenever there has been a tier table designated for the primary path, the records in any other tables manipulated at nodes along that path are constrained to be children of the record selected at the parent node. This includes manipulations of the basis table. This behavior is also true of table manipulations along non-primary paths, but does not include the basis table.

The node of interaction on the *primary* path, which is parent to the basis table, affects what SynApp2 refers to internally as providing the foreign key constraint. Whereas, any nodes on *non-primary* paths (but still technically have master/parent relationship with the basis table) are referred to internally as providing independent foreign keys.

In the case of the Ordered Book application, used in examples throughout this document, the `orders` table constrains the `ordered_book` table. The `book` table is manipulated independently. The notion of *constraint* and *independence* are trappings of the view domain and embraced by the page flow as a matter of SynApp2 framework design choices.

In all cases, before you can add a record to a table, the values for all foreign keys must be established. The *goal* of SynApp2 is to make sure this happens. The way it happens depends upon the tier table selection(s) you make, if any, in Page Flow panel of SynApp2 PageGen.

- When you designate a tier table along a path, you're going to get a set of forms that bear on a node of interaction and enable you to search for and select a record that will supply a foreign key value to your basis table.
- For any path that does not have a tier table designation, a `<select>` (i.e. drop-down) list is presented by the *Input Form*, associated with your basis table, allowing you to choose an `<option>` (i.e. record) that will supply the needed foreign key value.

The number of records you expect to be in a table is very often the determining factor of whether you designate a tier table for a given path.

- If you expect few records to be in a table, especially a fixed set of lookup items, a drop-down list will probably be suitable. Do not designate such a table as a tier table.
- If you expect many records to be in a table, say of customers, designate that table as a tier table. That will provide a search form with tabular record browsing and selection.

The order in which foreign key columns are defined (in your database schema) is a very important factor for the page flow. If a table has multiple foreign keys, consider which relationship represents the most defining, significant or natural access handle for a group of detail records to be associated, or constrained, to a single master record. Define that key column before any others.

In the case of the Ordered Book sample we've been using, records in the `ordered_book` table are most naturally grouped by `orders`. The records in the `orders` table are most naturally grouped by `customer`. While the `orders`, and `ordered_books` tables each have other foreign key columns, they really don't define the overall character of the database.

It's fast and easy to experiment with PageGen. See how your pages work when you do and do not designate tier tables.

While you're developing, there is a hypertext link mechanism, at the bottom of each page, which helps you quickly jump back and forth between the pages you're working on and the web application generator. The link is only visible when you login and select an AppID (or database) from the SynApp2 pages. Your users will not see the link.

Forms

Forms are automatically laid out on SynApp2 generated pages. They have labels and fields, or headings and columns, as is appropriate for their purpose. Various forms may be used for data input, output or both. Most forms have a control bar with buttons appearing as distinctive icons, related to their intended use. The state of the buttons, enabled or disabled, is reflected by their appearance. A full color button is enabled and a gray button is disabled. Buttons for certain actions may not appear at all, if the related action is not *authorized* for the currently logged in user.

As is the case for SynApp2 generated pages, request parameters that represent data are posted as one or more name value pairs, retrieved from the `<input>` elements of a form. A SynApp2 form is analogous to an HTML *form*, but does not actually use the `<form>` element, or the traditional `submit` mechanism. This has plusses and minuses.

SynApp2 forms don't appear within the DOM element hierarchy *as forms*, or support traditional validation techniques that involve the `onsubmit` event handler. But, they do cleanly avoid some pitfalls with regard to markup correctness when and as they are used with `XMLHttpRequest` objects, i.e. AJAX. SynApp2 forms don't need a dummy `action` attribute to be valid. Because the traditional forms submit mechanism is not used, it would have to be protected from inadvertent invocation, if compliant HTML 4.01 strict `<form>` elements were used. SynApp2 avoids these issues altogether, while strictly adhering to the W3C standard.

[Aside: SynApp2 will move forward with newer standards. But, careful attention to detail applied at this level of evolution provides firm footing.]

There are a number of functions/methods, implemented in `_shared_/synapp2.js`, that are more closely related to the view than to the *controller* domain – discussed later. Aside from drag and drop, the pattern for extending framework functionality isn't accomplished with JavaScript, so a detailed reference or discussion of the code will be left for another time and place. But more generally, there are some notable mechanisms and functions worth talking about.

All elements of the various GUI forms serve to display application data, at one point or another. The application data is mapped into the form elements by *name*. The form elements have

name attributes that correspond to *axis* names, set out in an XML response returned, by the server. The axis names are, or are derived from, column names defined in the database tables. There is a class, named `csx` – for Client Side eXchange – that performs the transfer of values from the XML to the HTML elements.

The value exchange happens during calls to the `update_container()` method. The `csx` object transparently handles exchanges, whether receiving *container* is form-like or table-like. Form-like means: has `<label>` and `<input>` elements. Table-like: means has (mostly) `<tr>`, `<th>` and `<td>` elements and may possibly have many rows.

There are two important functions than manage the display elements, `resize_container()` and `show_container()`.

The `resize_container()` function literally resizes the underlying `<table>` to have at least one row, but often more, usually depending on the number of data rows in the *payload* of the incoming responses, up to the value of the `LIMIT_ROWS` setting established (currently from the application's `custom.inc.php` file). The smaller of the two values is used, and is determined during each exchange/transfer.

The `resize` function uses the first row as a template for any additional rows, and creates *document object model* (DOM) elements as needed. The styling of the rows uses that same *class* and *style* attributes as the template row, with some minor adjustments to *even* numbered rows to achieve alternating shades. *Event handlers* are also attached to cells. Specific keyboard and mouse events are processed to provide shortcuts to important functionality. There will be further discussion of this later.

The `show_container()` function, along with a few helpers, manages a state-machine that hides or shows various forms as appropriate for each point (node of interaction) in the page *flow*. The mechanism is short, sweet and solid. It's worthy of study and understanding, and we'll leave you to do just that.

There is also a mechanism to manage the visible state of various `<button>` elements and to gate access to associated events (action triggers) like *search*, *add*, *edit*, *delete*, *close*, and *select*. The view receives tokens from an underlying authorization mechanism that can be customized to allow or disallow functionality according to login username.

Actions are gated at both the GUI and at the functional level (on the server), according a common set of directives. The details are covered by the [SynApp2 Customization](#) document. Take some time to study the design and implementation these mechanisms.

There's a boatload of stuff, built into every SynApp2 generated application, that manages a mountain of small, but important details, and it's all stuff that you don't have to find, invent or incorporate. It's just there. Go ahead and use it.

Form Roles

On a typical page, generated by SynApp2, there are one or more sets of forms. All of the forms in a set are tailored to the work with same table. As you might expect, each kind of form in a set plays a different *role* and becomes active according to the interactive page flow.

The set of form roles is:

- Search
- Display
- Select
- Input
- Statistics
- Report

Each role is supported by a different form definition and the markup for every form appears in the source code of the page. Each form has an `id` attribute, unique to the page, which is used by several mapping mechanisms to tie the form into the interactive page flow and data exchange cycles.

The page flow is an important consideration as you develop an application. The structure of the database – relations – drives the flow. As the developer you can control the flow. Perhaps the most important other variable affecting the page flow is the order of foreign key column definitions. The `ordered_book` example, seen throughout this document, would have a very different character if `ordered_book.id_book` occurred before `ordered_book.id_orders` in the table definition for `ordered_book`.

Another significant factor of page flow is determined by your selection, or not, of any tier table(s) from the page flow panel of the SynApp2 PageGen page.

The screenshot shows the SynApp2 PageGen application window. The title bar reads "SynApp2 PageGen". Inside the window, there are several tabs: "KeyMap", "Options", "PageGen", "Reports", and "Logout". The "PageGen" tab is active. Below the tabs, there are two input fields: "AppID (or database):" with the value "dbmain" and a green checkmark, and "QueryID (or table):" with a dropdown menu showing "ordered_book".

The main content area is divided into two panels. The left panel is titled "Page Settings - dbmain.ordered_book". It contains the following fields and controls:

- Basis Page:
- Basis Page Template: with a blue checkmark
- Report Page:
- Report Page Template: with a blue checkmark
- Buttons: "Run PageGen" and "Regenerate All:" with a checkbox.
- Page Status section with a text box containing:
Basis Page file: [ordered_book.htm](#) (2010-03-25 15:14:48)
Report Page file: [ordered_book.report.htm](#) (2010-03-25 15:14:48)

The right panel is titled "Page Flow". It displays a hierarchical diagram of the page flow. The diagram consists of a grid of boxes. The top row contains "gender (id_gender)". The second row contains "honor_title (id_honor_title)". The third row contains four boxes: "customer (id_customer)", "order_mode (id_order_mode)", "publisher (id_publisher)", and "genre (id_genre)". The fourth row contains "orders (id_orders)" and "book (id_book)". The fifth row contains "(FK1)" and "(FK2)". Below these are two "Clear" buttons. At the bottom of the panel is a box labeled "ordered_book (id)".

Selections in Page Flow panel

Compare the Page Flow panel to the generated page and consider the relationship between the tier table selections and the various forms.

Clear checkboxes on the Page Settings panel to prevent SynApp2 from generating or updating *Basis* or *Report* pages. If a page has already been created, and you want to exclude it from the application entirely, delete it (outside of SynApp2). If you want the page (or report), but don't want it to appear as a navigation (or report) tab, you can *omit* or selectively *authorize* its availability. A discussion about how to do this appears later.

Ordered_book

Book Customer Genre Ordered Book Reports Logout

Search Form - Customer

First Name:

Last Name:

City:

State:

Zipcode:

Phone:

Customer

Fill in values and then click Search

Search Form - Book

ISBN:

Publisher:

Title:

Synopsis:

Book

Fill in values and then click Search

Display Form - Orders

Order Number:

Ordered On:

Promised By:

Select Form - Book

| | ISBN | Publisher | Genre | Title | Synopsis | Price (retail) |
|--------------------------|------|-----------|-------|-------|----------|----------------|
| <input type="checkbox"/> | | | | | | |

Book

Navigate records, select a record and then click Edit (or click Add to create a new record)

Select Form - Ordered Book

| | Book | Quantity | Each | Discount | Amount |
|--------------------------|------|----------|------|----------|--------|
| <input type="checkbox"/> | | | | | |

Subtotal:

Tax:

Grand Total:

Ordered Book

Navigate records, select a record and then click Edit (or click Add to create a new record)

Page Layout corresponds to Page Flow selections in SynApp2 PageGen

The appearance of fields/columns on various forms is controlled from the SynApp2 Options page.

SynApp2 Options

SynApp2

KeyMap Options PageGen Reports Logout

AppID (or database): QueryID (or table):

Column Display Options - dbmain.ordered_book

| | Search Form | Display Form | Select Form | Input Form | Report Form | Report Column |
|-----------------------------------|--|--|--|---|--|--|
| id_orders INT 10 (FK1) | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> Align: -- default -- | Omit: No Editor: -- default -- Size: <input type="text"/> Rows: <input type="text"/> | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> Align: -- default -- Format: <input type="text"/> |
| id_book INT 10 (FK2) | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> Align: -- default -- | Omit: No Editor: -- default -- Size: <input type="text"/> Rows: <input type="text"/> | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> Align: -- default -- Format: <input type="text"/> |
| quan_ordered INT 10 | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> Align: Right | Omit: No Editor: -- default -- Size: <input type="text"/> Rows: <input type="text"/> | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> Align: Right Format: <input type="text"/> |
| price_retail DECIMAL 14 | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> Align: Right | Omit: No Editor: -- default -- Size: <input type="text"/> Rows: <input type="text"/> | Omit: No Size: <input type="text"/> | Omit: No Size: <input type="text"/> Align: Right Format: <input type="text"/> |

Process Options

The appearance of fields/columns is managed from the SynApp2 Options page

Customization settings from the SynApp2 Options page are stored in the `synapp2.inc.php` file, located in the application directory. There are addition customizations that can be introduced from the `custom.inc.php` file for the application. Once again, you'll find more information about this in the [SynApp2 Customization](#) document.

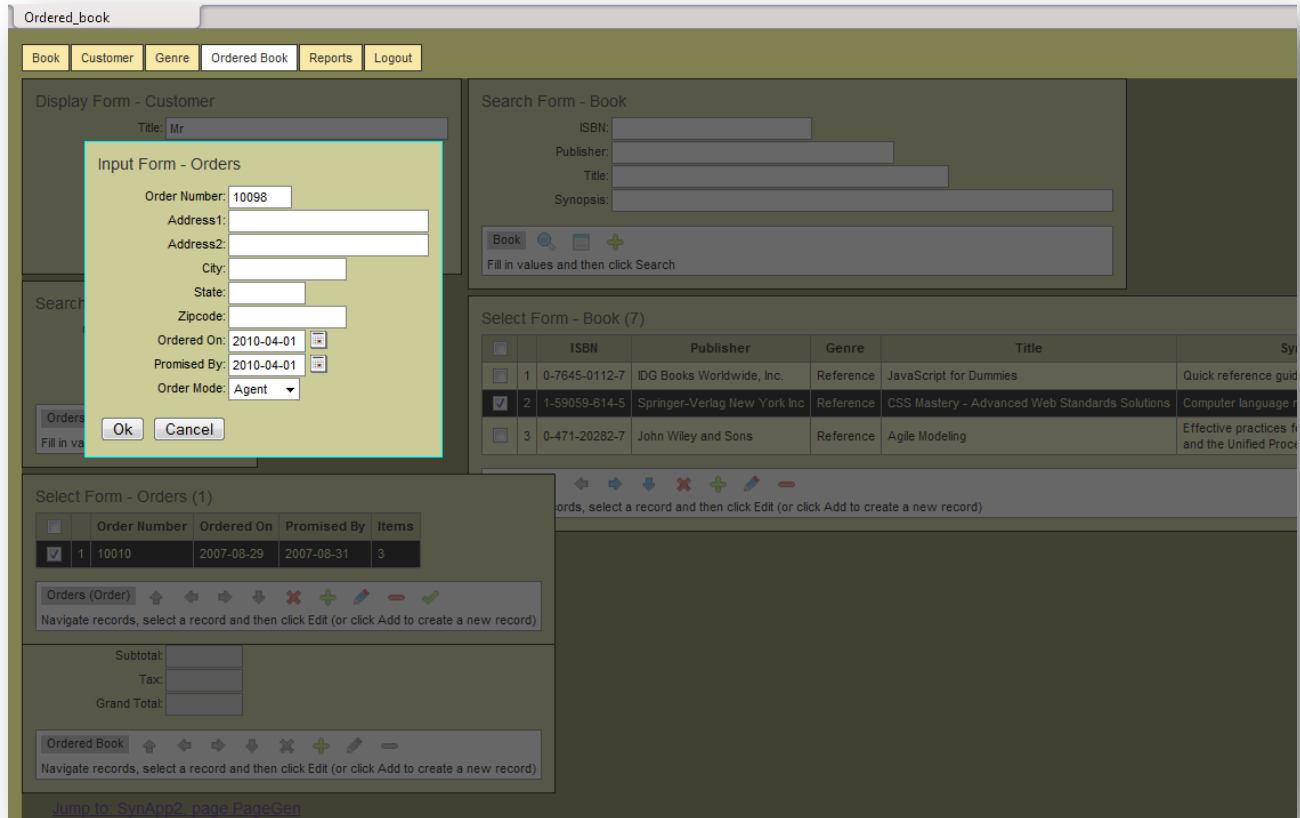
Each form has a corresponding *form key* that appears literally, and is frequently incorporated into variable names, throughout SynApp2 source code.

The set of form keys is:

- SFORM (**S**earch)
- DFORM (**D**isplay)
- TFORM (**s**elec**T**)
- IFORM (**I**ntput)
- AFORM (**s**tAtistics)
- FFORM (**r**eport **F**ilter)

Screen Image

A semi-transparent *Screen image* is used to overlay all but one or two active forms. The overlay will automatically adjust in size as changes are made to the window dimensions. Only the form that requires attention is accessible.



The Input Form is active (on top of the screen)

The screen image is incorporated with the following element:

```
<div id="id_iform_screen"><!-- empty --></div>
```

The default screen image container element is appears in `synapp2/template.htm` and is styled with CSS elements you'll find in `_shared/standard.css`.

Display Size and Page Layout

SynApp2 generated pages have a fluid layout, controlled by CSS. The default layout and styling of pages is defined in `_shared/standard.css`, and incorporated through a `<link>` element in `synapp2/template.htm`.

The default stylesheet link for SynApp2 and generated applications is:

```
<link rel="stylesheet" type="text/css" href="../_shared/standard.css">
```

Pages that deal with a hierarchy of tables may have two or more side-by-side columns of forms in the page flow. If your window or display is too narrow, the columns of forms will collapse into a single column. This isn't fatal, but it's not optimum.

Ordered_book

Book Customer Genre Ordered Book Reports Logout

Display Form - Customer

Title: Mr
 First Name: Mark
 Last Name: Twain
 State: MO
 Zipcode: 63401
 Phone: 573-221-9010

Display Form - Orders

Order Number: 10010
 Ordered On: 2007-08-29
 Promised By: 2007-08-31

Search Form - Book

ISBN:
 Publisher:
 Title:
 Synopsis:

Book

Fill in values and then click Search

Select Form - Book (7)

| | ISBN | Publisher | Genre | Title | Synopsis | Price (retail) |
|-------------------------------------|---------------|------------------------------|-----------|--|---|----------------|
| <input checked="" type="checkbox"/> | 0-7645-0112-7 | IDG Books Worldwide, Inc. | Reference | JavaScript for Dummies | Quick reference guide | 13.00 |
| <input type="checkbox"/> | 1-59059-614-5 | Springer-Verlag New York Inc | Reference | CSS Mastery - Advanced Web Standards Solutions | Computer language reference | 35.00 |
| <input type="checkbox"/> | 0-471-20282-7 | John Wiley and Sons | Reference | Agile Modeling | Effective practices for Extreme Programming and the Unified Process | 36.95 |

Book

Navigate records, select a record and then click Edit (or click Add to create a new record)

Select Form - Ordered Book (3)

| | Book | Quantity | Each | Discount | Amount |
|-------------------------------------|--|----------|-------|----------|--------|
| <input checked="" type="checkbox"/> | 1 Agile Modeling (0-471-20282-7) | 1 | 34.99 | 1.96 | 34.99 |
| <input type="checkbox"/> | 2 CSS Mastery - Advanced Web Standards Solutions (1-59059-614-5) | 2 | 35.00 | 0.00 | 70.00 |
| <input type="checkbox"/> | 3 Bulletproof Web Design (0-321-34693-9) | 1 | 34.99 | 5.00 | 34.99 |

A Collapsed Page Layout

If the forms collapse, there are several things you can try to make the layout better. First, try to make the text size smaller. Ctrl- [minus] (or ctrl+ [plus]) will change the text size. You could also try to make some of the forms narrower by reducing the column (display) size, or omitting (the display of) some columns altogether. Use the SynApp2 Options page to do this.

Omitting non essential columns from *Select* forms saves horizontal space, and omitting fields/columns from *Search* and *Display* forms saves vertical space. You can reduce the number of LIMIT_ROWS on *Select* forms too. See the [SynApp2 Customization](#) document for details.

Another way to control the layout would be to set a fixed width for one of the main page elements (`template.htm`) to be something like:

```
<div id="id_page_content" class="class_page_content" style="width:2500px;">
```

Here the width is set with a style attribute, but you could do it with CSS (`standard.css`). Specifying a width will most likely result in a horizontal scrollbar at the bottom of the page, but it will keep the layout from collapsing. Experiment to find a width value that works well for you.

Ordered_book

Book Customer Genre Ordered Book Reports Logout

Display Form - Customer

Title: Mr
First Name: Mark
Last Name: Twain
State: MO
Zipcode: 63401
Phone: 573-221-9010

Search Form - Book

ISBN:
Publisher:
Title:
Synopsis:
Book
Fill in values and then click Search

Display Form - Orders

Order Number: 10010
Ordered On: 2007-08-29
Promised By: 2007-08-31

Select Form - Book (7)

| | ISBN | Publisher | Genre | Title | Synopsis | Price (retail) |
|-------------------------------------|---------------------|---------------------------|-----------|---|--|----------------|
| <input checked="" type="checkbox"/> | 4 978-0-596-10159-2 | O'Reilly Media, Inc. | Reference | JavaScript: The Definitive Guide, Fifth Edition | JavaScript reference guide, Web 2.0, computer programming language | 49.99 |
| <input type="checkbox"/> | 5 1-56592-257-3 | O'Reilly Media, Inc. | Reference | Mastering Regular Expressions | Understanding the how and why of regular expressions | 34.95 |
| <input type="checkbox"/> | 6 0-7645-3196-4 | IDG Books Worldwide, Inc. | Reference | Hip Pocket Guide to HTML 4 | Concise overview of the HTML language | 14.99 |

Book
Navigate records, select a record and then click Edit (or click Add to create a new record)

Select Form - Ordered Book (3)

| | Book | Quantity | Each | Discount | Amount |
|-------------------------------------|--|----------|-------|----------|--------|
| <input type="checkbox"/> | 1 Agile Modeling (0-471-20282-7) | 1 | 34.99 | 1.99 | 34.99 |
| <input type="checkbox"/> | 2 CSS Mastery - Advanced Web Standards Solutions (1-59059-614-6) | 2 | 35.00 | 0.00 | 70.00 |
| <input checked="" type="checkbox"/> | 3 Bulletproof Web Design (0-321-34093-9) | 1 | 34.99 | 5.00 | 34.99 |

Subtotal: 139.98
Tax: 10.85
Grand Total: 150.83

Ordered Book
Navigate records, select a record and then click Edit (or click Add to create a new record)

A Normal Page Layout

Use any combination of methods to make the page appear as it should.

Once a page is generated, it can become its own template. This is useful, as long as your changes do not appear between output (delimiter) tags. Use the page file name as the template file name. Do this after you have modified the existing page to incorporate a logo or other specialized markup.

From the SynApp2 PageGen - Page Settings panel, specify the Basis Page Template, relative to the `synapp2/synapp2` application directory, with the general form:

```
../<your_appid>/<your_page_name>.htm
```

A few SynApp2 output delimiter pairs from <install_dir>/synapp2/template.htm:

```
//<!--{xch}-->
//<!--{/xch}-->

//<!--{map}-->
//<!--{/map}-->

//<!--{ovl}-->
//<!--{/ovl}-->

<!--{css}-->
<!--{/css}-->

<!--{group}-->
<!--{/group}-->

<!--{iform}-->
<!--{/iform}-->
```

If you don't want your markup to be overwritten, it must be outside any delimiter pairs. If you must put markup inside the delimiters, clear the appropriate checkbox on the SynApp2 PageGen – Page Settings panel. This will prevent SynApp2 from updating (writing to) the file containing your specialized markup.

If you make changes to your database structure or to SynApp2 Options that should be reflected in the page markup, you're either going to need to re-introduce your customizations, or propagate updated elements from an alternate page, generated under/with a different name.

About Browsers

All elements of the display are designed to scale nicely when the browser text size changes. This works especially well with Firefox. Internet Explorer has text size control too, but it's available only from the browser command menu. The ctrl+ and ctrl- shortcut keys behave differently, depending on browser. IE7+ performs an unfortunate graphic zoom effect where as Firefox, and others, seem to adjust the document elements more sensibly. They all behave differently.

Overall Safari works pretty well, on all platforms, even on the iPhone , iPod Touch and iPad. Google Chrome works nicely, but doesn't really have a useful status bar. Versions of Opera prior to 10 just weren't up to the task. Version 10.1 works reasonably well, but is slightly out of step with the others when it comes to keyboard event handling. It's serviceable though. But, if the page content exceeds the window height (the vertical scroll bar is present), you may see the display scroll as you use the up and down cursor keys in SynApp2 tabular *Select* forms.

Keep features like forms *auto-fill* and *auto-capitalization* turned *off* until you are familiar enough with how the application should work, to know if there are any issues raised. Do watch for issues caused by overzealous caching of the web pages, any time you regenerate pages, make sure you do a hard refresh for each of the affected pages (ctrl-R or ctrl-F5 depending on browser), or clear the browser cache altogether. In any case, if you experience any quirks that detract, consider using the Mozilla Firefox browser to run your applications.

Controller

The **controller** is comprised of elements executed on both the server and the in the client browser. The top-level object on the server side of the controller is implemented in the PHP source file: `_shared_/action.php`. The other side of the controller coin is implemented in the JavaScript source file: `_shared_/synapp2.js`. Even though elements of the controller and view implementations coexist, the domains remain clearly separated.

Exchange Cycle

Ordinary application data moves back and forth between user interface (UI) elements, visible in your web browser, and a data repository on your web server according to events that you, as a user, initiate. The exchange technique is commonly referred to as AJAX.

The exchange cycle consists of a triggering event, followed by parameter extraction and action encoding, sending of a [HTTP] request, receiving of a request, parameter and action decoding, action processing, result processing, response formatting, return of a [XML] response, response reception. Any response data is subsequently transferred to the view with a clean row-name-value interface.

Requests convey several fundamental parameters along with ordinary application data.

The *appid* – application ID – is the most significant request parameter. In the parlance of software terminology, *appid* bears some similarity to the concept of a namespace. The *appid* provides a context within which other request parameters are interpreted. Within the context of an *appid*, a *qid* – query ID – typically provides sub-context for actions and is a symbolic reference used as a key to select a specific interrogation or process that is then used to affect the data model.

Customization of the exchange cycle is easily accomplished, but by default, a number of standard behaviors should be understood. We'll touch on a few elements that are important to the foundation of understanding how SynApp2 works.

The heart of SynApp2 is a Structured Query Language (SQL) generator. The elements of an SQL statement are assembled from a variety of sources and submitted to the database engine. This will all be covered in more detail, but for now, let's focus on how *appid* and *qid* affect things.

As you hopefully know, the relational database engines that support SynApp2, conceptually store data in tables. A group of related tables are, again conceptually, considered to be a database. That's what MySQL calls it. Oracle is different in that it collects tables (and other objects) into a schema. SynApp2 casts the Oracle schema back into the notion of a database.

As far as SynApp2 is concerned, a database has a name. If you don't specify otherwise, the value of *appid* corresponds directly to a database name. It also provides the name for the *application directory*, where SynApp2 generated pages are stored. This can be changed, of course, but we'll not concern ourselves with that here.

Given a database name, SynApp2 will cause the database engine to *connect* to it. The topic of database connections will also be deferred. Here, we'll just acknowledge that a connection happens. Once connected to a database, SQL queries typically refer to one or more tables.

If you don't specify otherwise, the value of a *qid* is the name of a table. This too can be changed, or *customized*, but we'll leave that for later.

Virtually every influence over application (and framework) behavior, keys off of *qid* values. Different aspects of influence occur according to a small set of *action* and *mode* parameters as they vary during the course of interaction. Request actions are processed in light of the other request parameters.

The *qid* abstraction is unique to SynApp2 and provides outstanding separation from and between implementation details and factors governing inputs, control logic, processing, and outputs. All points of programmatic interaction are defined by distinct combinations of *qid*, *action* and *mode* values. Input data, supplied by various means, is uniformly presented during exchanges between the client browser and the server, and is managed according to the specific situation expressed.

All SQL statements originate on, and in the protected environment of the server. All data values and query terms are escaped and checked, to prevent SQL insertion attacks.

All aspects of the applications and databases are protected by an access controller. No data can be retrieved or affected without proper authentication and authorization.

A SynApp2 page will have a set of [HTML] forms for the basis table. The page may also have sets of forms for related tables. Each set of forms is mapped to a *qid*, and thereby, to a table. There are several specialized kinds of forms that may be part of a set and different ones may be active (and visible) at different points along various exchange cycles and interactive *flow* of the page.

When a specific form becomes active, it serves as a *container*. A container can provide both a source and a destination for ordinary application data, and indirectly, provides a source of request parameters by way of mapping between the container *id* attribute and a *qid*. Other parameter values, related to a container, are similarly mapped.

Given the following parameters:

```
appid="MyData",  
qid="Greetings",  
greeting="Hello World!",  
action="ACTION_INSERT";
```

-- you could expect something like this to occur on your web server --

```
INSERT INTO MyData.Greetings (greeting) VALUES ("Hello World!");
```

To complete the exchange cycle, a response is always returned for each request. The response will typically include status or error information and ordinary application data.

Events

The controller is driven by events that begin when your web browser visits a SynApp2 generated page.

Page Load

During the page loading process, various HTML elements are processed. JavaScript code is executed and this results in several objects and internal data structures being instantiated.

The SynApp2 application ID, or AppID (appid), and Page ID (pid) are defined at this time.

```
set_appid('dbmain');  
set_pid('ordered_book');
```

Associations between `id` attributes of HTML *container* elements, SynApp2 query ID's (qid), and role in the page flow, are accomplished by registration methods that appear in the `/{map}--> (map)` section of a SynApp2 generated page.

Once the SynApp2 page load completes, control is passed to a handler specified by the value of the `onload` attribute that appears in the HTML tag `<body>`.

```
<body id="id_body" onload="page_init();">
```

The `page_init()` function invokes several other methods, which in-turn fire several events that exchange data between your browser and your web server, using an AJAX mechanism.

```
function page_init()  
{  
    do_init();  
    do_app_nav('id_app_nav');  
}
```

The functions perform miscellaneous internal chores, including dynamic generation of the navigation tabs that appear at the top of the page. The `forms_reset()` method literally resets HTML form elements and causes the input focus to settle on an appropriate `<input>` element.

The `do_init()` method initializes the state of the user interface elements and page flow mechanism. It is only required for SynApp2 generated pages. It may be omitted for pages created by other means and for other purposes, although it doesn't hurt to include it.

The `do_app_nav()` method ties the page in with the navigation tab mechanism. The function argument `'id_app_nav'` is the container where the dynamically generated elements will be inserted. If you create a supplemental page you'll need to make an entry in `custom.inc.php` to make it appear as one of the navigation tabs. See below.

For supplemental custom pages, that you create manually, there are several other useful

functions that you may want to call from the `page_init()` function.

Examples of these include:

```
map_vkey_action('id_myform', VKEY_ENTER, my_submit_func, true);
set_input_text_value('id_start_date', getStartDate());
set_focus('id_end_date');
```

Hopefully, the purpose and usage of these functions are somewhat self evident. You'll find them implemented in `synapp2.js` and used extensively. Take a look in `_shared_/synapp2.js` for examples. You'll find the complete set of virtual key (VKEY) definitions there.

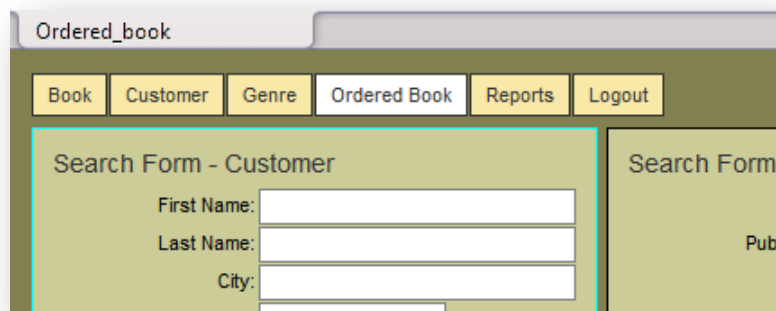
On SynApp2 generated pages, you'll see a *cyan colored border* around the form that is listening for keyboard shortcuts supported by the VKEY mechanism.

TODO: table of shortcuts vs. form roles

Shortcut key behaviors depend on the form role.

Page Navigation and Report Tabs

The navigation tabs at the top of pages are dynamically generated in response to an event triggered during `page_init()` by a call to `do_app_nav()`.



Navigation tabs for the Ordered_book application

The active tab appears *white*, according to CSS elements you'll find in `_shared_/standard.css`.

When a navigation tab is clicked, the browser loads a new page, via an ordinary hypertext link, and the event sequence of the new page proceeds accordingly. You can customize your application by *including* supplemental tabs, or by *omitting* others. You can selectively *authorize* (i.e. display) tabs based upon login username.

A good way to get started with a custom page (or report) is to copy one of page files generated with SynApp2 PageGen and then make changes to that copy. Use an interactive page or a report page as a base, according to the flavor of page you're going to make.

Page navigation tabs, report tabs and authorization entries are all made in the

custom.inc.php customization file in the application directory.

Examples of adding page navigation tabs:

```
$this->m_data[APPID]['iso_training'][INCL][NAV] = 'training, attendance, roster';
```

```
$this->m_data[APPID]['mfg_scheduling'][INCL][NAV][] = array(A_HREF=>'scheduling',  
A_TEXT=>'<em><strong>Build - Schedule</strong></em>');
```

You can add report tabs the same way:

```
$this->m_data[APPID]['purchasing'][INCL][RPT] = 'product, supplier';
```

All tab names imply filenames, e.g., product.htm, roster.htm, scheduling.htm, etc.

The page/report file *must exist on the server* in order to appear on a tab. If the file isn't there, the tab won't be either. This is useful behavior. If you find that a page or report, especially one for a support table, contributes to tab clutter, just delete the associated page and/or report and the tab(s) will go away.

Use NAV or RPT OMIT entries when you want to keep the page or report (file), but don't want a tab for it. You can access any page directly by entering its URL or from a browser shortcut. Access control to all pages and reports, is always available through the authorization mechanism.

You can control availability of various application elements with authorization entries in the custom.inc.php file.


Examples of authorization entries:

```
$this->m_$admin_users = 'richard,tracy';  
$principal_users = ',manny,moe,jack';  
  
$this->m_data[APPID]['projtrack'][AUTH_QID]['activity'] = $admin_users;  
  
$this->m_data[APPID]['projtrack'][AUTH_PID]['activity'] = $admin_users;  
//$this->m_data[APPID]['projtrack'][AUTH_PID]['activity_projects'] = $admin_users .  
$principal_users;  
$this->m_data[APPID]['projtrack'][AUTH_PID]['worker'] = $admin_users;  
$this->m_data[APPID]['projtrack'][AUTH_PID]['project'] = $admin_users;  
$this->m_data[APPID]['projtrack'][AUTH_PID]['project_status'] = $admin_users .  
$principal_users;  
$this->m_data[APPID]['projtrack'][AUTH_PID]['process'] = $admin_users;  
$this->m_data[APPID]['projtrack'][AUTH_PID]['reports'] = $admin_users .  
$principal_users;  
  
$this->m_data[APPID]['projtrack'][AUTH_RID]['status_type'] = $admin_users;  
$this->m_data[APPID]['projtrack'][AUTH_RID]['activity'] = $admin_users .  
$principal_users;  
  
$this->m_data[APPID]['projtrack'][QID]['worker'][AUTH_ADD] = $admin_users;  
$this->m_data[APPID]['projtrack'][QID]['worker'][AUTH_EDIT] = $admin_users;  
$this->m_data[APPID]['projtrack'][QID]['worker'][AUTH_DELETE] = $admin_users;
```

Any *application, query, page, report* or *data action* that does not have a specific `AUTH_XXXX` entry, will be accessible to any login username.

See the [SynApp2 Customization](#) document for details.

Search

Most page interactions begin with a *Search Form*. Search is initiated with the  button on the control bar of the form. The form serves as a *filter* for the request. The underlying action is `ACTION_SELECT` and the mode is `MODE_GET_NORM`. See `_shared_/synapp2.js` for the full set of request action and mode definitions. The request action and mode parameters are posted to `_shared_/action.php`. Look there to see how these values drive the framework *controller*.

All values of the `<input>` elements of the *Search Form* are posted along with the request parameters. As the response is handled, the associated *Select Form* (container) is updated with any retrieved data rows/records.

Values from the `<input>` elements of Search Forms are used to build the `WHERE` clause of the generated `SELECT` statement. The `LIKE` operator is combined with non-empty name-value pairs. The `%` wild card character appears at either end of each literal value, and the sub-expressions evaluate as `TRUE` if the literal appears anywhere in the column value. This produces a record filtering effect as it is applied.

```
select ... where publisher.pub_name like '%reilly%' and
          book.book_title like '%java%' ...
```

The search event and most other table navigation events have handlers similar to:

```
function do_search(tform_id, sform_id)
{
    var xch = new cxl();

    xch.set_container(tform_id);
    xch.set_filter(tform_id, sform_id);
    xch.send_request();
}
```

Event handlers for the various form actions are assigned according to control names and are bound to controls through the `onclick` attribute. Keyboard and mouse shortcuts are subsequently bound to whatever is assigned to `onclick`.

Named control elements from a SynApp2 generated page:

```
...

<div class="class_control_bar_controls">
<button type="button" name="do_search" title="Search"></button>
```

```

<button type="button" name="do_reset" title="Reset"></button>
<button type="button" name="do_add" title="Add"></button>
</div><!-- class_control_bar_controls -->

...

```

Binding controls to handlers:

```

...

switch (control_name)
{
case 'do_search':
    eval("control_element.onclick = function() { do_nav_first( ... ); }");
    map_vkey_action(container_id, VKEY_ENTER, control_element.onclick);
    break;


...

```

The search event is tied into the framework using the same techniques as the events described hereafter. As we've set out a number of the important implementation details for the search event, the discussions for the remaining events will be less verbose.





There are some subtleties that simply can't be explained more concisely than the code that implements them. You're encouraged to explore the JavaScript code in `_shared_/synapp2.js` with a debugger. All of the major browsers have interactive debuggers. Set a few breakpoints, step through some functions, and discover how the framework behaves.

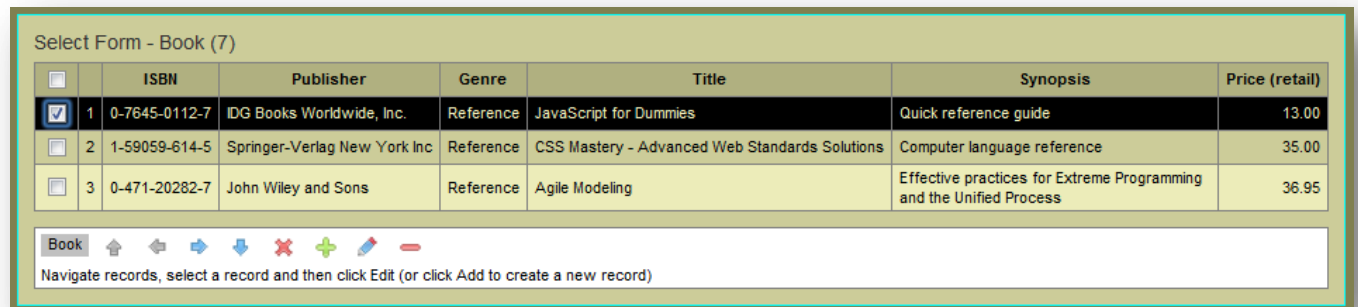
Reset

A reset event is initiated with the  button on the control bar of a *Search Form*. The reset event triggers the clearing of all *Search Form* `<input>` elements, followed by a search event as described above.








Browse (Row/Record Navigation)

Any records that satisfy the search filter (criteria) are presented in a tabular *Select Form*. If the number of records retrieved exceeds the maximum number of rows allowed for the form (controlled by a `LIMIT_ROWS` entry in `custom.inc.php` - the default value is 5), then controls and keyboard shortcuts are enabled to facilitate paginated row/record browsing.

Button controls, to trigger navigation events for *first page* , *page up/previous* , *page down/next*  and *last page* , are available on the control bar of the *Select Form*. Their enabled or disabled state is appropriately managed according record/row position.



| | ISBN | Publisher | Genre | Title | Synopsis | Price (retail) | |
|-------------------------------------|------|---------------|------------------------------|-----------|--|---|-------|
| <input checked="" type="checkbox"/> | 1 | 0-7645-0112-7 | IDG Books Worldwide, Inc. | Reference | JavaScript for Dummies | Quick reference guide | 13.00 |
| <input type="checkbox"/> | 2 | 1-59059-614-5 | Springer-Verlag New York Inc | Reference | CSS Mastery - Advanced Web Standards Solutions | Computer language reference | 35.00 |
| <input type="checkbox"/> | 3 | 0-471-20282-7 | John Wiley and Sons | Reference | Agile Modeling | Effective practices for Extreme Programming and the Unified Process | 36.95 |

Book       

Navigate records, select a record and then click Edit (or click Add to create a new record)

The first three (of seven) rows are visible with only page down/next or last page navigation available

A page down event triggers scrolling to a new page in the tabular *Select Form*:



| | ISBN | Publisher | Genre | Title | Synopsis | Price (retail) | |
|-------------------------------------|------|-------------------|---------------------------|-----------|---|--|-------|
| <input checked="" type="checkbox"/> | 4 | 978-0-596-10199-2 | O'Reilly Media, Inc. | Reference | JavaScript: The Definitive Guide, Fifth Edition | JavaScript reference guide, Web 2.0, computer programming language | 49.99 |
| <input type="checkbox"/> | 5 | 1-56592-257-3 | O'Reilly Media, Inc. | Reference | Mastering Regular Expressions | Understanding the how and why of regular expressions | 34.95 |
| <input type="checkbox"/> | 6 | 0-7645-3196-4 | IDG Books Worldwide, Inc. | Reference | Hip Pocket Guide to HTML 4 | Concise overview of the HTML language | 14.99 |

Book       

Navigate records, select a record and then click Edit (or click Add to create a new record)

After Page Down is triggered, rows four through six (of seven) are visible with all row navigation choices available


As you might expect, some keyboard shortcuts are usually available. If your workstation has them, the *Home*, *PgUp*, *PgDn* and *End* keys correspond to and trigger the four cardinal row/record navigation events. *Up Arrow* and *Down Arrow* keys move the row highlight and will trigger page up/down navigation events from the top or bottom row of the *Select Form*.

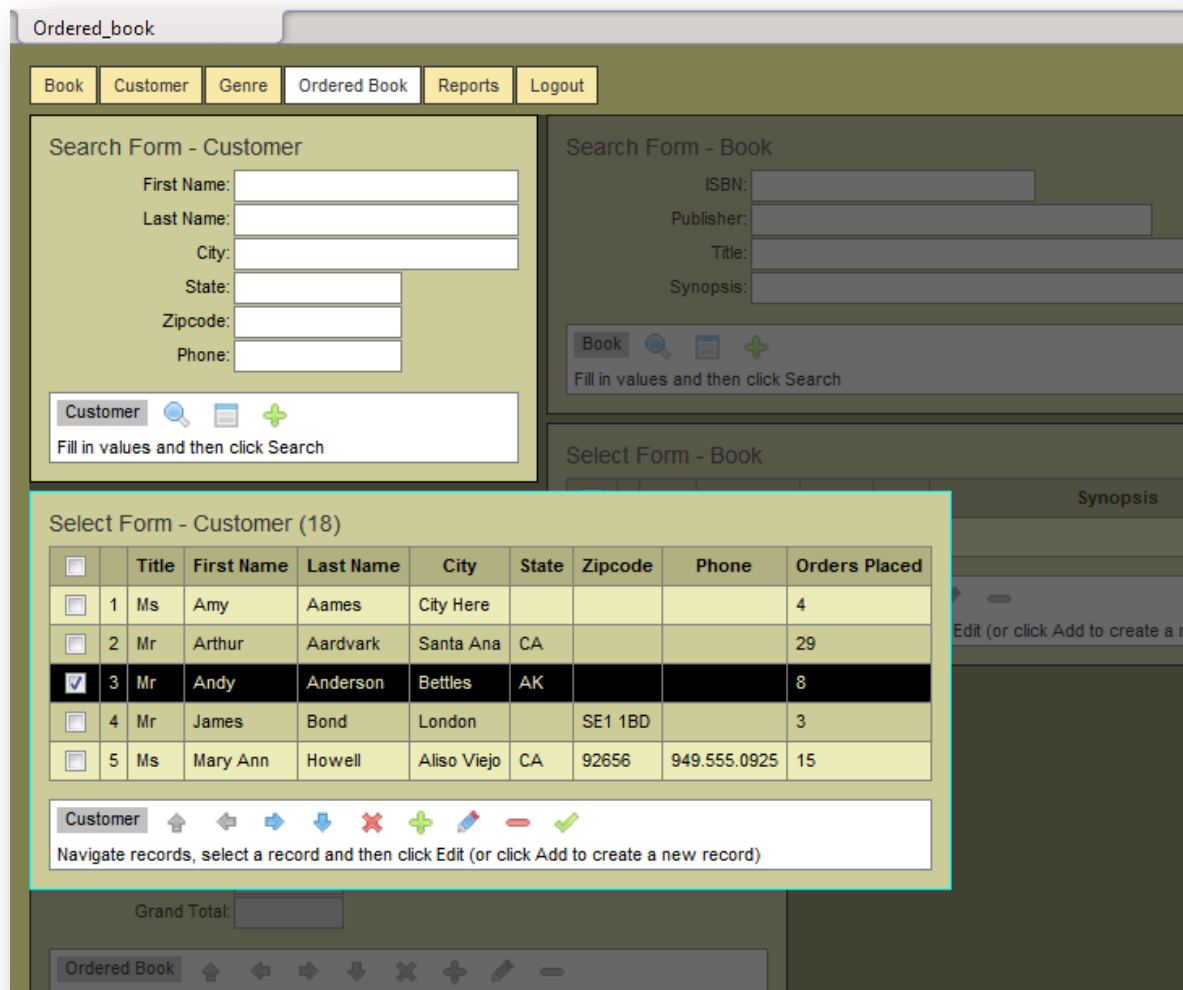
For each navigation event, the underlying action is `ACTION_SELECT` and the mode is `MODE_GET_NORM`. As each response is handled, the *Select Form* (container) is reset/cleared, resized (as needed), and updated with retrieved data rows/records (if any).

Whenever a *Select Form* is paired (appears) with *Search Form*, row/record navigation is filtered by any values that appear in the fields of the *Search Form*.

Select

The select event is only needed when, and therefore only triggered from (a) *Select Form(s)* involved with a hierarchy of related tables that are manipulated on the same page SynApp2 generated page.

The select event, initiated with the  button, on the control bar of a *Select Form*, triggers a request where the underlying action is `ACTION_SELECT` and the mode is `MODE_GET_NORM`. The container is a *Display Form* which gets updated with values from the selected row/record. The *Select Form*, from which the select event was triggered, is rolled up (hidden) and replaced by the *Display Form*.



Ordered_book

Book Customer Genre Ordered Book Reports Logout

Search Form - Customer

First Name:



Last Name:

City:

State:

Zipcode:

Phone:

Customer  

Fill in values and then click Search



Search Form - Book

ISBN:

Publisher:

Title:

Synopsis:

Book  










Fill in values and then click Search

Select Form - Book

Synopsis









Select Form - Customer (18)

| | Title | First Name | Last Name | City | State | Zipcode | Phone | Orders Placed |
|-------------------------------------|-------|------------|-----------|----------|-------------|---------|--------------------|---------------|
| <input type="checkbox"/> | 1 | Ms | Amy | Aames | City Here | | | 4 |
| <input type="checkbox"/> | 2 | Mr | Arthur | Aardvark | Santa Ana | CA | | 29 |
| <input checked="" type="checkbox"/> | 3 | Mr | Andy | Anderson | Bettles | AK | | 8 |
| <input type="checkbox"/> | 4 | Mr | James | Bond | London | SE1 1BD | | 3 |
| <input type="checkbox"/> | 5 | Ms | Mary Ann | Howell | Aliso Viejo | CA | 92656 949.555.0925 | 15 |

Customer         

Navigate records, select a record and then click Edit (or click Add to create a new record)

Grand Total:

Ordered Book        

A follow-up event (and request) is generated. The underlying action is again `ACTION_SELECT` and the mode is `MODE_GET_NORM`. The container is a *Select Form* associated with a table that is on the *many* side of a relation. The rows/records, if any, returned in the response are constrained to be only those related to (details/children of) the *one* (master/parent) record that was selected.

Ordered_book

Book Customer Genre **Ordered Book** Reports Logout

Display Form - Customer

Title: Mr

First Name: Andy

Last Name: Anderson

State: AK

Zipcode:

Phone:

Search Form - Book

ISBN:

Publisher:

Title:

Synopsis:

Book

Fill in values and then click Search

Search Form - Orders

Order Number:

Ordered On:

Promised By:

Orders (Order)

Fill in values and then click Search

Select Form - Book

| | ISBN | Publisher | Genre | Title | Synopsis |
|--|------|-----------|-------|-------|----------|
| | | | | | |

Book

Navigate records, select a record and then click Edit (or click Add to create a new record)


Select Form - Orders (8)

| | Order Number | Ordered On | Promised By | Items |
|-------------------------------------|--------------|------------|-------------|-------|
| <input checked="" type="checkbox"/> | 1 10052 | 2008-05-14 | 2008-05-16 | 3 |
| <input type="checkbox"/> | 2 10084 | 2009-08-20 | 2009-08-20 | 1 |
| <input type="checkbox"/> | 3 10085 | 2010-01-09 | 2010-01-09 | 4 |
| <input type="checkbox"/> | 4 10086 | 2010-01-09 | 2010-01-15 | 2 |
| <input type="checkbox"/> | 5 10087 | 2010-01-09 | 2010-01-09 | 1 |

Orders (Order)

Navigate records, select a record and then click Edit (or click Add to create a new record)

Many (8) Orders rows/records belong (are constrained) to one (1) selected Customer

The  button, on any *Select Form*, is not become enabled unless *exactly one* row/record, visible in the form, is checked (and therefore highlighted).

The *Enter* or *Right Arrow* keys also fire the select event if the *Select Form* is active (has the cyan border around it) and the select event/button is enabled.

The select event can also be initiated with a click on a *Select Form* row data cell (not the selection checkbox or the row number). Such a click will exclusively check/highlight that row and initiate the select event. This works even if one or more rows already checked/highlighted.

If the table containing the selected record is not the master/parent of the page *basis table*, a *Select Form* is presented along with its associated *Select Form*. The *Select Form* can be used to filter records in the associated *Select Form*. Whether or not *Search Form* is used, any records that are retrieved and displayed in the *Select Form*, are *constrained* to be details/children of the selected master/parent.

Sort rows/records with a click on any *Select Form* column heading. Click the same column again to reverse the order. Click in the upper left corner of the form to restore the default sort order.

A diamond symbol, next to a column heading, indicates alternate sort order and direction:

The screenshot shows a web application interface for a 'Customer' database. At the top, there's a tab labeled 'Customer'. Below it, a navigation bar contains buttons: 'Book', 'Customer', 'Genre', 'Ordered Book', 'Reports', and 'Logout'. The main content area is divided into two sections. The top section, titled 'Search Form - Customer', contains input fields for 'First Name:', 'Last Name:', 'City:', 'State:', 'Zipcode:', and 'Phone:'. Below these fields is a search bar with a magnifying glass icon and a green plus icon, with the text 'Fill in values and then click Search'. The bottom section, titled 'Select Form - Customer (18)', displays a table of customer records. The table has columns: 'Title', 'First Name', 'Last Name', 'City', 'State', 'Zipcode', 'Phone', and 'Orders Placed'. The first row is selected, showing a customer named Mr. Arthur Aardvark from Santa Ana, CA, with 29 orders placed. Below the table is a navigation bar with icons for navigating between records (up, down, left, right arrows) and a red 'X' icon, with the text 'Navigate records, select a record and then click Edit (or click Add to create a new record)'.


| | Title | First Name | Last Name | City | State | Zipcode | Phone | Orders Placed ♦ |
|-------------------------------------|-------|------------|-----------|----------|-------------|---------|-------|-----------------|
| <input checked="" type="checkbox"/> | 1 | Mr | Arthur | Aardvark | Santa Ana | CA | | 29 |
| <input type="checkbox"/> | 2 | Ms | Mary Ann | Howell | Aliso Viejo | CA | 92656 | 15 |
| <input type="checkbox"/> | 3 | Mr | Richard | Howell | Aliso Viejo | CA | 92656 | 9 |
| <input type="checkbox"/> | 4 | Mr | Andy | Anderson | Bettles | AK | | 8 |
| <input type="checkbox"/> | 5 | Ms | Amy | Aames | City Here | | | 4 |

Customers are sorted by descending number of orders placed

The default sort order for a table is by primary key (PK) value. As primary keys are numeric and typically assigned in a sequence of ascending values, the order of records is effectively chronological. The default sort order for a qid/table can be set with an `ORDER` entry in `custom.inc.php`.

Records are sorted as they are retrieved from the database – not by the browser.

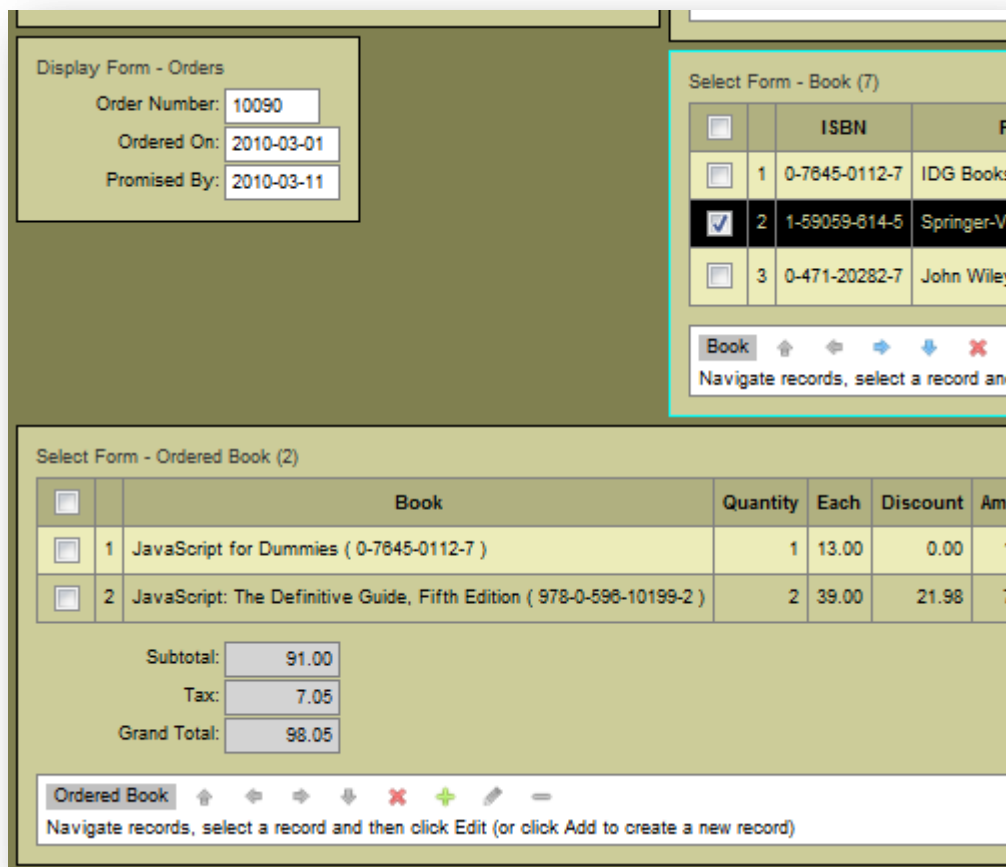
Add

The Add event is initiated with the  button on the control bar of either a *Search Form* or a *Select Form*. A request/response cycle is triggered and underlying action is ACTION_SELECT and the mode is MODE_GET_INIT. The container for the request is an *Input Form*.

As the response is handled, any returned column values are used to initialize fields of the *Input Form* and the form is presented. The seemingly innocuous Add event subsequently triggers a cascade of useful behaviors involved with form initialization.

There is a lot of information to convey about what happens as Add sequence unfolds and the application is set up to capture input and insert a new record into a table. Please take your time. Read carefully and consider how the various parts of the discussion would likely apply to similar situations you may have to deal with in your work.

The following discussion centers on typical situation you might find in an order-entry application. Here, a book is about to be added as a line-item of an order.



Display Form - Orders






Order Number: 10090

Ordered On: 2010-03-01

Promised By: 2010-03-11

Select Form - Book (7)

| | ISBN | P |
|-------------------------------------|------|---------------------------|
| <input type="checkbox"/> | 1 | 0-7645-0112-7 IDG Books |
| <input checked="" type="checkbox"/> | 2 | 1-59059-814-5 Springer-Ve |
| <input type="checkbox"/> | 3 | 0-471-20282-7 John Wiley |

Book     

Navigate records, select a record and









Select Form - Ordered Book (2)

| | Book | Quantity | Each | Discount | Amc |
|--------------------------|---|----------|-------|----------|-----|
| <input type="checkbox"/> | 1 JavaScript for Dummies (0-7645-0112-7) | 1 | 13.00 | 0.00 | 1 |
| <input type="checkbox"/> | 2 JavaScript: The Definitive Guide, Fifth Edition (978-0-596-10199-2) | 2 | 39.00 | 21.98 | 7 |

Subtotal: 91.00


Tax: 7.05

Grand Total: 98.05

Ordered Book        

Navigate records, select a record and then click Edit (or click Add to create a new record)

Here, the  (Add) button is enabled because of the checked/highlighted row in *Select Form - Book*

The  button, on any *Select Form* associated with the page the *basis table*, does not become enabled until a row/record in every *Select Form* associated with a master/parent table is chosen (checked/highlighted).

Form initialization, triggered by the Add event, is accomplished, in this case, with two exchange cycles. The second cycle is triggered because SynApp2 PageGen recognized the signature of a lookup/copy situation and automatically incorporated a call to `reg_lookup()` in the generated page. There's more on this in a moment.

The first exchange returns the default value for `ordered_book.quan_ordered`, as specified in the SQL table definition for `ordered_book`. Additional data is returned to initialize what SynApp2 calls *expanded elements*. And, there's more on expanded elements - shortly.

An initialization response with default values for an ordinary field and expanded elements:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<response>
<version>0.1.7</version>
<authentication>succeeded</authentication>
<authorization>succeeded</authorization>
<request_action>select</request_action>
<request_mode>get_init</request_mode>
<qid>ordered_book</qid>
<query></query>
<found_rows>0</found_rows>
<insert_offset>0</insert_offset>
<rows>1</rows>
<cols>1</cols>
<firstpage>1</firstpage>
<lastpage>1</lastpage>
<payload>
<axis>quan_ordered</axis>
<tr><td>1</td></tr>
</payload>
<select name="ordered_book_id_orders">
<option value="90">10090 2010-03-01 ( Anderson, Andy )</option>
</select>
<select name="ordered_book_id_book">
<option value="2">CSS Mastery - Advanced Web Standards Solutions ( 1-59059-614-5 )</option>
</select>
</response>
```

The SQL column sub-expressions for expanded elements are provided according to MACRO entries in `custom.inc.php`. See the [SynApp2 Customization](#) document for details.

A customization MACRO entry for the foreign key column `'ordered_book.id_book'`:

```
$this->m_data[APPID]['dbmain'][QID]['ordered_book'][MACRO]['ordered_book.id_book']  
= "concat( book.book_title, ' ( ', book.ISBN, ' ) ' )";
```

If there is no MACRO entry for a foreign key, SynApp2 automatically uses value the first non-key column according to the master/parent table definition, if there is one. If not, as a last resort, the raw key value is used.

Column expressions and names – *aliases* - for expanded elements are synthesized and incorporated into generated SQL statements and query results. The column alias `ordered_book_id_book` was formed from the (qualified) foreign key column name `'ordered_book.id_book'`, using `'_'` (underscore) rather than `'.'` (dot) to combine the terms. The resulting column alias is `'ordered_book_id_book'`. This convention is easy to remember.

When working with SynApp2 generated SQL and query results, there are always two columns for every foreign key involved. One column is the raw key value and name. The other is the expanded value with its synthesized alias, as described above.

After the Add event and initialization exchange, in this particular discussion, a second exchange occurs because the tables `book` and `ordered_book` each have an identically named column - `price_retail`. The value of `book.price_retail` is copied to `ordered_book.price_retail`.

If you wanted lookup/copy behavior between columns with dissimilar name, include a `FETCH` entry in `custom.inc.php`. As PageGen runs, the necessary call to `reg_lookup()` will be included in the generated page. This lookup/copy functionality is available when two tables have a master/parent – detail/child (one-to-many) relationship. If more than one column name is shared, additional exchanges occur as necessary until every column value has been copied.

If you don't want automatic column lookup/copy behavior to occur, don't have identically named columns.

Any lookup/copy exchange, triggered subsequent to the initial Add event, has an underlying action of `ACTION_SELECT` with the mode being `MODE_FETCH`. The container for the request is the *Input Form*.

A lookup/copy response:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>  
<response>  
<version>0.1.7</version>  
<authentication>succeeded</authentication>  
<authorization>succeeded</authorization>  
<request_action>select</request_action>
```

```

<request_mode>fetch</request_mode>
<qid>book</qid>
<query></query>
<found_rows>1</found_rows>
<insert_offset>0</insert_offset>
<rows>1</rows>
<cols>10</cols>
<firstpage>1</firstpage>
<lastpage>1</lastpage>
<payload>
<axis>id,isbn,id_publisher,book_id_publisher,id_genre,book_id_genre,book_title,
synopsis,price_wholesale,price_retail</axis>
<tr id="2"><td>2</td><td>1-59059-614-5</td><td>2</td><td>Springer-Verlag New York
Inc</td><td>4</td>
<td>Reference</td><td>CSS Mastery - Advanced Web Standards
Solutions</td><td>Computer language reference
</td><td>21.00</td><td>35.00</td></tr>
</payload>
<debug_msg>
$_POST
Array
(
    [_request_action_] => select
    [_request_mode_] => fetch
    [_request_pid_] => ordered_book
    [_request_qid_] => book
    [_request_appid_] => dbmain
    [_request_flow_] => Array
        (
            [0] => customer,1020,id_orders,orders,id_customer
            [1] => orders,90,id_orders,ordered_book,id_orders
            [2] => ordered_book,185
            [3] => book,2,id_book,ordered_book,id_book
        )

    [_request_context_] => Array
        (
            [0] => customer,1020
            [1] => orders,90
            [2] => ordered_book,185
            [3] => book,2
        )

    [_request_flow_basis_] => id_orders
    [_request_limit_offset_] => 0
    [_request_limit_rows_] => 1
    [_request_order_] =>
    [_request_pk_] => 2
    [_request_show_response_] => true
)
</debug_msg>
</response>

```

The <debug_msg> element (and contents) is not included in a routine exchange. It appears here (for illustration) because the request_show_response parameter was set.

After all initialization and lookup/copy exchanges are completed, the *Input Form* is presented - ready for interactive editing.

The *Input Form* waits for either of two events associated with buttons:

The screenshot shows a software interface for managing book orders. A modal dialog titled "Input Form - Ordered Book" is centered on the screen. It contains three input fields: "Book" (containing "CSS Mastery - Advanced Web Standards Soluti"), "Quantity" (containing "1"), and "Each" (containing "35.00"). Below these fields are "Ok" and "Cancel" buttons. The background interface is dimmed but visible. It includes a "Display Form - Customer" section with fields for Title, First Name, Order Number, Ordered On, and Promised By. To the right is a "Search Form - Book" with fields for ISBN, Publisher, Title, and Synopsis. Below the search form is a table of search results with columns for checkboxes, a numeric index, ISBN, and Publisher. At the bottom of the background interface is a "Select Form - Ordered Book (2)" section containing a table with columns for checkboxes, a numeric index, Book title, Quantity, Each, Discount, and Amount. Below this table are fields for Subtotal and Tax.

| Book | Quantity | Each | Discount | Amount |
|---|----------|-------|----------|--------|
| 1 JavaScript for Dummies (0-7645-0112-7) | 1 | 13.00 | 0.00 | 13.00 |
| 2 JavaScript: The Definitive Guide, Fifth Edition (978-0-596-10199-2) | 2 | 39.00 | 21.98 | 78.00 |

Subtotal: 91.00
Tax: 7.05

The Input form is displayed with fields initialized and focus set to first editable field

The field labeled Book is read-only as it is an expanded element.

While the *Input Form* waits, lookup/copy (FETCH) request/response cycles may be triggered by onSelect events of <select> (i.e. drop-down) list elements present on the *Input Form*. The use of a *Search Form* / *Select Form* pair, as opposed to a list-box is controlled by a choice on the Page Flow panel of SynApp2 PageGen.

An *Input Form* with a list-box to choose a book:

Ordered_book

Book Customer Genre Ordered Book Reports Logout

Display Form - Customer

Title: Mr

Input Form - Ordered Book

Book: Agile Modeling (0-471-20282-7)

Quantity: 1

Each: 36.95

Ok Cancel

Display Form - Customer

Order Number: 10090

Ordered On: 2010-03-01

Promised By: 2010-03-11

As a book is selected, `book.price_retail` is copied to `ordered_book.price_retail`:

Ordered_book

Book Customer Genre Ordered Book Reports Logout

Display Form - Customer

Title: Mr

Input Form - Ordered Book

Book: CSS Mastery - Advanced Web Standards Solutions (1-59059-614-5)

Quantity: 1

Each: 35.00

Ok Cancel

Display Form - Customer

Order Number: 10090

Ordered On: 2010-03-01

Promised By: 2010-03-11

Regardless of how a book record is chosen, the same lookup/copy mechanism is used to propagate the `retail_price` value from the book record to `ordered_book` record.

The *Ok* button triggers a request where underlying action is `ACTION_INSERT` and the mode is `MODE_PUT_NORM`. The *Input Form* is the container for the request. The *ENTER* key can also trigger *Ok*.

If the entered data is successfully *validated*, a new record is inserted.

If any entry fails validation, a descriptive message appears for each condition. All of the *Input Form* values are *sticky*. The *Input Form* again waits for an *Ok* or *Cancel* button click.

The screenshot shows a web application interface. In the background, there is a 'Display Form - Customer' with fields for 'Title' (Mr) and 'First Name' (Andy). To the right is a 'Search Form - Book' with fields for 'ISBN', 'Publisher', 'Title', and 'Synopsis'. In the foreground, an 'Input Form - Ordered Book' dialog box is open. It contains three input fields: 'Book' (CSS Mastery - Advanced Web Standards Soluti), 'Quantity' (0), and 'Each' (free). Above these fields, two red error messages are displayed: 'Quantity must be greater than or equal to '1'' and 'Each must be a valid 'DECIMAL''. At the bottom of the dialog box are 'Ok' and 'Cancel' buttons. Below the dialog box, a 'Promised By' field shows the date '2010-03-11'.

Messages for two data validation failures

If the *Cancel* button is clicked, a request/response cycle is triggered and underlying action is `ACTION_SELECT` and the mode is `MODE_CANCEL`. The container for the request is the *Input Form*. The *ESC* key can also trigger *Cancel*.

Once a record is inserted, or the *Cancel* button is clicked, another request/response cycle is triggered to synchronize the associated *Select Form* with the new record or whatever row/record that had been selected, prior to the Add event that started things rolling. The underlying action is `ACTION_SELECT` and the mode is `MODE_GET_NORM`. The container is the associated *Select Form*. The cycle is almost identical to one triggered by *Search*.

Ordered_book

Book Customer Genre Ordered Book Reports Logout

Display Form - Customer

Title: Mr

First Name: Andy

Last Name: Anderson

State: AK

Zipcode:

Phone:

Display Form - Orders

Order Number: 10090

Ordered On: 2010-03-01

Promised By: 2010-03-11

Search Form - Book

ISBN:

Publisher:

Title:

Synopsis:

Book

Fill in values and then click Search

Select Form - Book (7)

| | ISBN | Publisher | Genre | Title | Synopsis | Price (retail) |
|-------------------------------------|-----------------|------------------------------|-----------|--|---|----------------|
| <input type="checkbox"/> | 1 0-7845-0112-7 | IDG Books Worldwide, Inc. | Reference | JavaScript for Dummies | Quick reference guide | 13.00 |
| <input checked="" type="checkbox"/> | 2 1-59059-614-5 | Springer-Verlag New York Inc | Reference | CSS Mastery - Advanced Web Standards Solutions | Computer language reference | 35.00 |
| <input type="checkbox"/> | 3 0-471-20282-7 | John Wiley and Sons | Reference | Agile Modeling | Effective practices for Extreme Programming and the Unified Process | 38.95 |

Book

Navigate records, select a record and then click Edit (or click Add to create a new record)

Select Form - Ordered Book (3)

| | Book | Quantity | Each | Discount | Amount |
|-------------------------------------|--|----------|-------|----------|--------|
| <input checked="" type="checkbox"/> | 3 CSS Mastery - Advanced Web Standards Solutions (1-59059-614-5) | 1 | 35.00 | 0.00 | 35.00 |

Subtotal: 128.00

Tax: 9.77

Grand Total: 135.77

Ordered Book

Navigate records, select a record and then click Edit (or click Add to create a new record)

Select Form - Ordered Book is synchronized with the newly inserted row/record

Edit

The Edit event is initiated with the button on the control bar of a *Select Form*. A request/response cycle is triggered and underlying action is ACTION_SELECT and the mode is MODE_GET_NORM. The container for the request is an *Input Form*.

The button, on any *Select Form*, does not become enabled until *exactly one* row/record, visible in the form, is chosen (checked/highlighted).


The *Input Form* is initialized with values from the response (that were retrieved from the database table, for the selected record).


From this point in the page flow, the *Input Form* waits exactly as it does when triggered by the Add event. The only difference is an action value of ACTION_UPDATE, if the *Ok* button is clicked.

While the *Input Form* is waiting, the lookup/copy mechanism for list-boxes is in play, just as is when the event is Add. See the pullout discussion of the lookup/copy mechanism above.

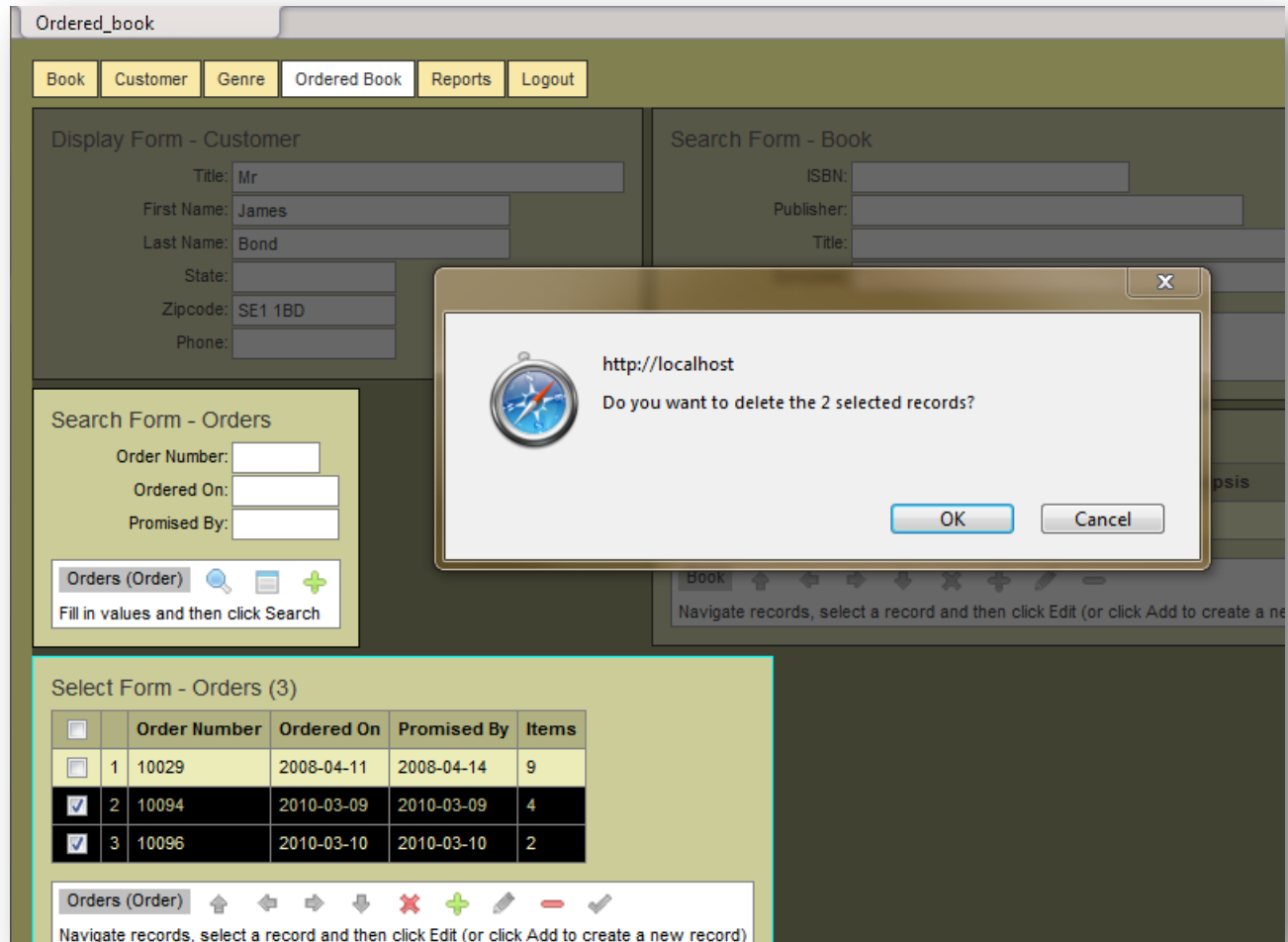
If validation is successful, the selected record is updated and the associated *Select Form* is synchronized with the updated record.

Delete

The Delete event, initiated with the  button on the control bar of a *Select Form*

The  button, on any *Select Form*, does not become enabled until at least *one* (or more) row(s)/record(s), visible in the form, is/are chosen (checked/highlighted).

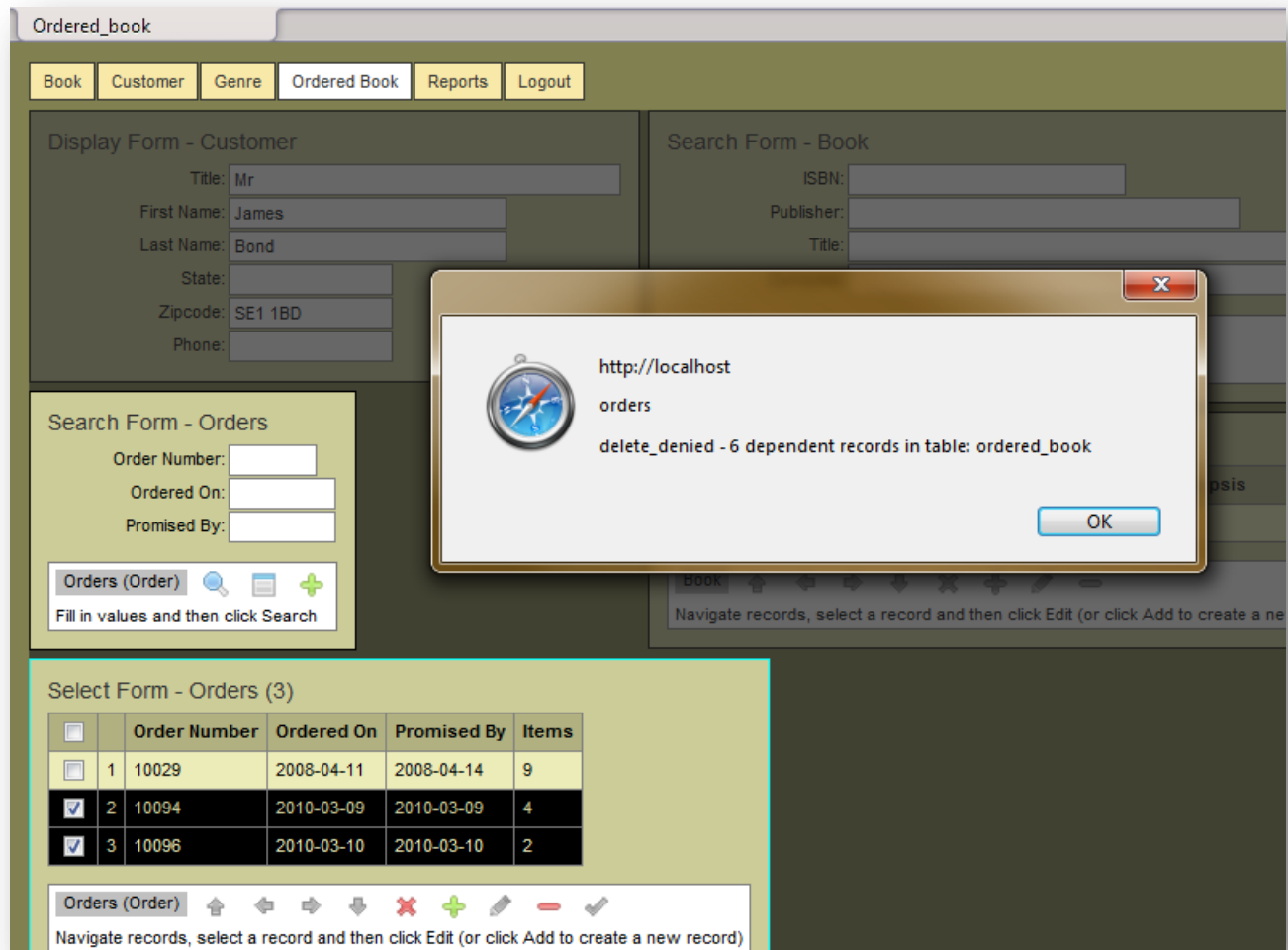
A confirmation dialog appears in response to the Delete event.



To carry out record deletion, you must confirm the operation by clicking OK

If you Cancel, the confirmation dialog is dismissed without further action.

If you continue, by clicking Ok, an exchange cycle is triggered with an underlying action of ACTION_DELETE and mode value of MODE_PUT_NORM. There is no container for the request.




The SynApp2 framework has detected record dependencies and will not allow the selected records to be deleted

The framework automatically checks for record dependencies. If any detail/child records for any of the selected master/parent records are found, a message dialog is presented. Otherwise, the selected rows/records are deleted and the associated *Select Form* is synchronized to the first row with *another* exchange cycle.

The SynApp2 framework safeguards the referential integrity of the database and will prevent orphan records from occurring. Cascade delete is not currently supported, so you must delete any detail/child record(s) before you can delete a master/parent record.

You can easily create and integrate custom processes to do any combination of periodic record maintenance chores, such as: backup, export, purging, or data warehousing.

Close

The Close event, initiated with the  button on the control bar of a *Select Form*. A request/response cycle is triggered where the underlying action is ACTION_SELECT and the mode is MODE_CLOSE.

As the response is handled, the active form will shift to the *Select Form* associated with a master/parent table, or to the *Search Form* of the current *Search / Select Form* pair, depending on the conditions. Forms are dismissed and presented as appropriate.

The Close event allows you to work backward through the table/form hierarchy of the page. The resulting behavior is essentially the inverse of what happens for the Select event.

If a *Select Form* is active, close can be triggered with the ESC key.

Statistics

After any actions that do, or could, change records in a table, additional exchange cycles *may* be triggered, ostensibly to allow computation, or even post-processing of dependent values or data, and optionally displaying the results in elements of a *Statistics Form*.

Typical customization entries for statistics:

```
$subtotal = "sum(ordered_book.quan_ordered * ordered_book.price_retail)";
$tax = "{$subtotal} * 0.0775";

$this->m_data[APPID]['sample'][QID]['ordered_book'][FETCH]['subtotal'] =
"format({$subtotal}, 2)";
$this->m_data[APPID]['sample'][QID]['ordered_book'][FETCH]['tax'] =
"format({$tax}, 2)";
$this->m_data[APPID]['sample'][QID]['ordered_book'][FETCH]['grand_total'] =
"format({$subtotal} + {$tax}, 2)";
```

Given the above entries, SynApp2 PageGen will emit the following to ordered_book.htm:

```
//<!--{map}-->

...

reg_blind('id_aform_00_ordered_book_subtotal');
reg_blind('id_aform_00_ordered_book_tax');
reg_blind('id_aform_00_ordered_book_grand_total');

...

//<!--{/map}-->

...

<div id="id_aform_00_ordered_book" class="class_form_std class_form_stat">
<label>Subtotal:</label>
<input readonly id="id_aform_00_ordered_book_subtotal" type="text"
name="subtotal" size="10" style="text-align:right;"><br>
<label>Tax:</label>
```

```
<input readonly id="id_aform_00__ordered_book__tax" type="text" name="tax"
size="10" style="text-align:right;"><br>
<label>Grand Total:</label>

...
```

The `reg_blind()` statements set up a map that is used to prevent *Stat Form* `<input>` values from being posted as parameters to the request.

The *Statistics Form*, or *Stat Form*, elements appear within a dedicated `<div>` of a *Select Form*. Request/response cycles are automatically triggered, once for each element. The underlying action is `ACTION_SELECT` and the mode is `MODE_FETCH`. The *Stat Form* is the container for the response.

The exchange applies a corresponding `FETCH` expression, keyed to the request *qid* and selected by *name*.

The screenshot displays the 'Ordered_book' application interface. It features a navigation bar with tabs: Book, Customer, Genre, Ordered Book, Reports, and Logout. The main content area is divided into several sections:

- Display Form - Customer:** Contains input fields for Title (Mr), First Name (James), Last Name (Bond), State, Zipcode (SE1 1BD), and Phone.
- Display Form - Orders:** Contains input fields for Order Number (10098), Ordered On (2010-03-10), and Promised By (2010-03-10).
- Search Form - Book:** Contains input fields for ISBN, Publisher, Title, and Synopsis, with a 'Book' button and a 'Fill in values and then click Search' instruction.
- Select Form - Book (7):** A table listing books with checkboxes for selection. The table has columns for ISBN, Publisher, Genre, and a fifth column (partially visible as 'Java').
- Select Form - Ordered Book (2):** A table listing ordered books with checkboxes for selection. The table has columns for Book, Quantity, Each, Discount, and Amount.

Below the 'Select Form - Ordered Book (2)' table, there is a summary section with the following values:

- Subtotal: 109.89
- Tax: 8.52
- Grand Total: 118.41

At the bottom of the interface, there is a navigation bar with a 'Book' button and a 'Navigate records, select a record and then click Edit (or click Add to create a new record)' instruction.

Statistics Form elements appearing within *Select Form - Ordered Book*

By default, *Statistics Form* values are right-aligned. If you want to change text alignment or adjust field display size use additional customization entries.

Typical customization entries for text alignment and column size:

```
$this->m_data[APPID]['sample'][QID]['ordered_book'][AFORM]['tax'][COL_ALIGN] =  
ALIGN_L;  
  
$this->m_data[APPID]['sample'][QID]['ordered_book'][AFORM]['tax'][COL_SIZE] =  
'5';
```

Note the `AFORM` form key that associates the entries with the *Statistics Form*. See the [SynApp2 Customization](#) document for information about column output formatting.

Templates

Templates supply static markup and code and define insertion points for markup and code generated by SynApp2 PageGen. You can add or change markup in a template. That's how you would add a logo or banner text or change the background. You can incorporate third party JavaScript libraries too.

If not present, files for `index.html` and `<appid>.welcome.htm` are generated into the application directory (by SynApp2 PageGen) only when Regenerate All is *not* checked. If you need to create or want to recreate those pages, clear the Regenerate All checkbox and run PageGen for any individual page that belongs in your application.

In the `_shared_` directory, you'll find template files for the index and welcome pages, plus `template.htm`, used for application pages, and `template.report.htm`, used for report pages. You'll also find `login.htm` and `standard.css`. You can modify the files to personalize you applications.

For the app and report templates, you can make specialized versions and choose which to use, from the PageGen - Page Settings panel. See the earlier discussion about how an existing application page can act as a template.

Reports

Tabular reports are generated as `PDF` files and presented by way of *Adobe Reader*. The reader is almost universally installed, and if not, the report file may be downloaded and saved for later use.

Customizations of reports can include specification of `EXTRA` columns and statistics with `DETAIL_SUMMARY_COLS` entries in `custom.inc.php`. You can also control page size and orientation. Any column widths that s that don't have an explicit specification, are dynamically calculated – including multi-row cells.

A *Report Form* is presented to allow control over scope of the records to be reported. As describe earlier, values entered in the fields are incorporated into the SQL using the `LIKE` operator, except for dates. Dates are entered as ranges and a date picker control is accessible. Leave a date field blank for an open ended query.

Reports

Lot Widget Operation Resource Profile Cursors Daily Schedule Op Type Wip Build - Capabilities Build - Shifts

Lot Widget Operation Resource Capability Shift Profiles Work Shift Schedule Ops Wip Daily Schedule

Report Form - Shift Profiles

Work Shift:

Shift Date >= : 2010-01-31

<= : 2010-02-07

Profile:

Shift Capacity:

Download Export: No

Shift Profiles

Fill in values and then click Report

A Report Form

Report data can be downloaded to a file or opened directly by programs such as Excel.

Your reports can have different headings and formats – this one has cell borders turned on:

Reports

Lot Widget Operation Resource Profile Cursors Daily Schedule Op Type Wip Build - Capabilities Build - Shifts Build - WIP Build - Schedule Rollback Reports Logout

Lot Widget Operation Resource Capability Shift Profiles Work Shift Schedule Ops Wip Daily Schedule

1 / 2 100% Find

Shift Profiles - 7 Apr 2010 (Wed) - 18:27 US/Pacific

| Shift Date | Shift Name | Profile | Shift Capacity |
|-------------|-----------------------|--------------------|----------------|
| 01 Feb 2010 | Monday - Day Shift | AL (Normal) | 480 |
| | | CM (Normal) | 480 |
| | | FG (Normal) | 480 |
| | | HL (Normal) | 240 |
| | | JK (Normal) | 480 |
| | | JL (Normal) | 480 |
| | | JN (Normal) | 480 |
| | | LL (Normal) | 480 |
| | | MM (Normal) | 480 |
| | | POST CURE (Normal) | 480 |
| | | PS (Normal) | 480 |
| | | RL (Normal) | 480 |
| | | SHIPPING (Normal) | 480 |
| | | TT (Normal) | 480 |
| | | VH (Normal) | 480 |
| | | YL (Normal) | 480 |
| 02 Feb 2010 | Tuesday - Day Shift | AL (Normal) | 480 |
| | | CM (Normal) | 480 |
| | | FG (Normal) | 480 |
| | | HL (Normal) | 240 |
| | | JK (Normal) | 480 |
| | | JL (Normal) | 480 |
| | | JN (Normal) | 480 |
| | | LL (Normal) | 480 |
| | | MM (Normal) | 480 |
| | | POST CURE (Normal) | 480 |
| | | PS (Normal) | 480 |
| | | RL (Normal) | 480 |
| | | SHIPPING (Normal) | 480 |
| | | TT (Normal) | 480 |
| | | VH (Normal) | 480 |
| | | YL (Normal) | 480 |
| 03 Feb 2010 | Wednesday - Day Shift | AL (Normal) | 480 |
| | | CM (Normal) | 480 |
| | | FG (Normal) | 480 |
| | | HL (Normal) | 240 |
| | | JK (Normal) | 480 |
| | | JL (Normal) | 480 |

A tabular PDF report displayed in Adobe Reader

Custom Processes

Customized processing functions can be defined. Elements of the SynApp2 exchange cycle can be harnessed to support interactive forms used for gathering parameters and invoking processes.

Almost anything can be done using this mechanism. See the [SynApp2 Customization](#) document for more information about custom processing.

The screenshot shows the 'Build - Shifts' application window. The top menu bar includes: Lot, Widget, Operation, Resource, Profile, Cursors, Daily Schedule, Op Type, Wip, Build - Capabilities, Build - Shifts (selected), Build - WIP, Build - Schedule, Rollback, Reports, and Logout.

The main form area is titled 'Build Shifts' and contains the following fields:

- Beg: 2010-01-31
- End: 2010-02-07
- Shift: Day
- Duration: (empty)
- Profile: Normal
- Capacity: (empty)

Below the form are buttons: Process, Set / Update, Clear, Resource, Day, Sun, Mon, Tue, Wed, Thu, Fri, Sat.

The 'Daily Schedule, Shift Profiles' section contains a table with the following data:

| Resource \ Shift Date | Sunday 1/31 | Monday 2/1 | Tuesday 2/2 | Wednesday 2/3 | Thursday 2/4 | Friday 2/5 | Saturday 2/6 | Sunday 2/7 |
|-----------------------|-------------|----------------|----------------|----------------|----------------|----------------|----------------|------------|
| AL | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| CM | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| FG | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| HL | | Day Normal 240 | Day Normal 240 | Day Normal 240 | Day Normal 240 | Day Normal 240 | Day Normal 240 | |
| JK | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| JL | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| JN | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| LL | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| MM | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| POST CURE | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| PS | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| RL | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| SHIPPING | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| SM | | Day - off | Day - off | Day - off | Day - off | Day - off | Day - off | |
| TT | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| VH | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |
| YL | | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | Day Normal 480 | |

Custom GUI to drive shift schedules is neatly integrated into a sophisticated application

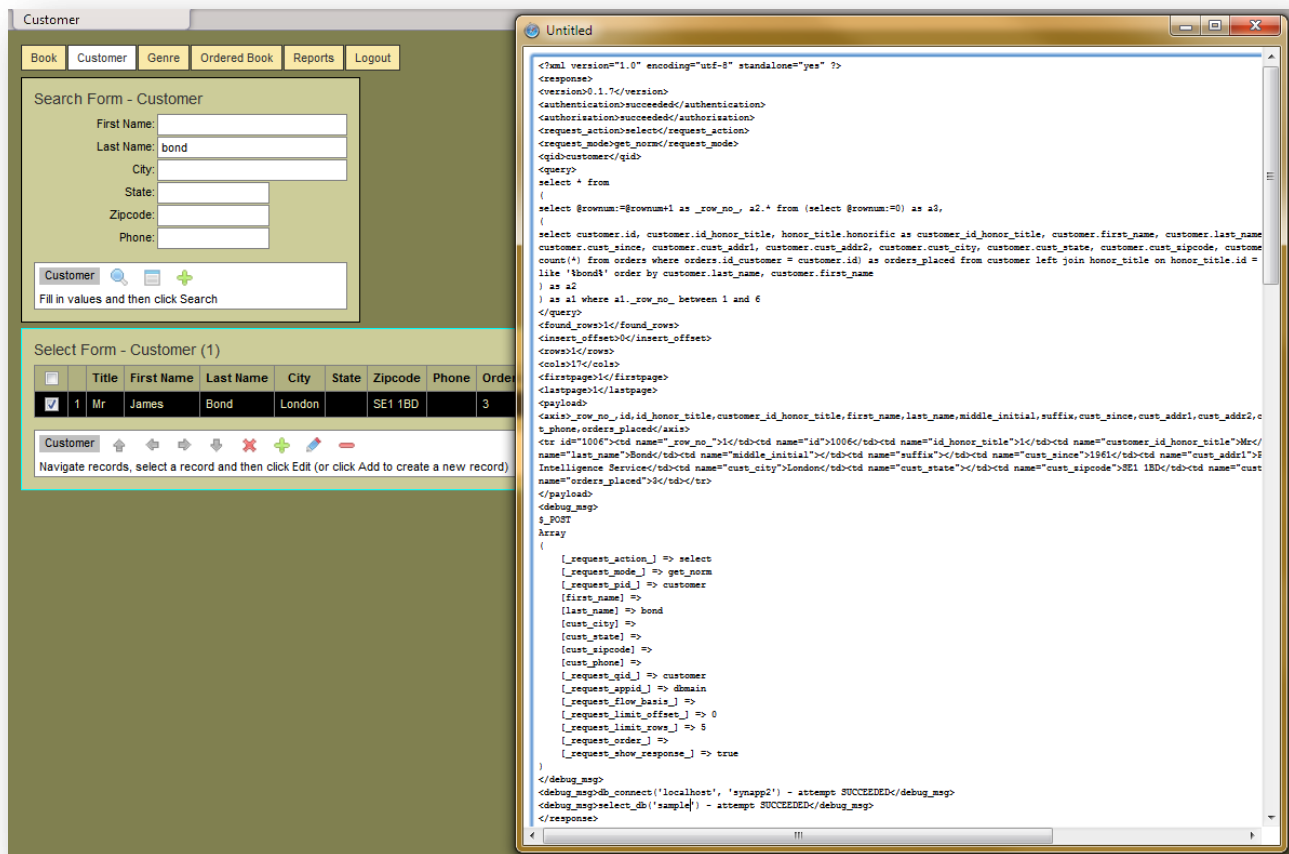
Your function(s) can generate markup that gets returned and inserted as the `innerHTML` of an *adhoc* container.

Debug Message Window

You're almost certainly going to find it useful to monitor the exchange cycle as you develop and test the pages of your applications.

To cause the Debug Message Window to appear, append "?show_response=1" to the URL of your SynApp2 generated page, in the address field of your browser, and reload the page.

IMPORTANT: You'll need to enable pop-ups in your web browser in order to see the message window. You should be able to do this, just for your web server, by listing it as an exception.



The Debug Message Window showing the response triggered by a Search event

The first time the message window pops up; it will appear on top of the application window. Subsequent messaging will be reflected, but not cause the window to appear on top. This behavior can be changed by adjusting a variable in the source file: `_shared_/synapp2.js`.

```
var m_debug_msg_window_to_front = false; //true; // DEFAULT: change as appropriate
```

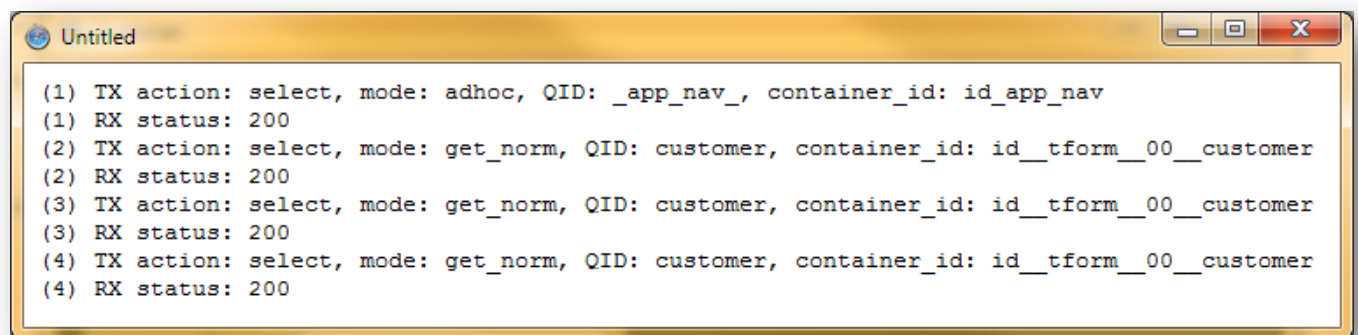
By default, the debug message window is not forced to appear on top because it can interfere with the page element focus behavior. It's generally better to arrange windows so there is no overlap.

You can also use a browser add-on to display exchange details, DOM and CSS elements and to perform interactive JavaScript source code debugging. The [Firebug](#) add-on for Mozilla Firefox browser is excellent.

If you want to see what's going on with the PHP side of the framework, you can use the function `add_debug_msg($var_or_expression, 'text_label')`. This will dump values into the response and (open) update the debug message window.

Event Message Window

You may find it useful to monitor the events driving the exchange cycles as you develop and test the pages of your applications.



The triggering of every exchange cycle is logged in the Event Message Window

To cause the Event Message Window to appear, edit `_shared_/synapp2.js`. Search for the variable `m_event_msg_is_enabled` variable and set its value to `true`.

```
var m_event_msg_is_enabled = true;
```

The first time the event message window pops up; it will appear on top of the application window. Subsequent messaging will be reflected, but not cause the window to appear on top. Arrange your windows as needed, to see the messages.

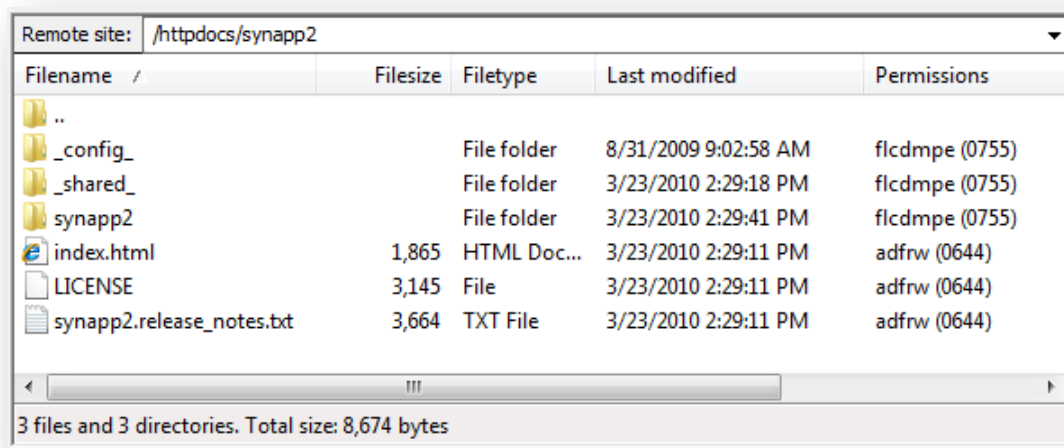
Hard Error Reporting

Under normal circumstances, hard errors should not occur. If they *do* happen, the debug message window will appear with data about the failure. Again, if the message window is already open, it won't necessarily appear on top. See the discussion about the debug message window above.

Installation and Deployment

The footprint of a SynApp2 installation is extremely small. Configuration requirements are minimal and completely segregated. The SynApp2 framework and any applications can be updated without disturbing the configuration. This makes maintenance super fast and super easy.

Whether you're updating SynApp2 or your working applications, you can do it in seconds – just copy a tiny handful of files to your production server. With (free) FTP software like FileZilla, it's a one-step drag and drop operation to deploy newly generated applications or updates on a remote server.



| Filename | Filesize | Filetype | Last modified | Permissions |
|---------------------------|----------|-------------|----------------------|----------------|
| .. | | | | |
| _config_ | | File folder | 8/31/2009 9:02:58 AM | flcdmpe (0755) |
| _shared_ | | File folder | 3/23/2010 2:29:18 PM | flcdmpe (0755) |
| synapp2 | | File folder | 3/23/2010 2:29:41 PM | flcdmpe (0755) |
| index.html | 1,865 | HTML Doc... | 3/23/2010 2:29:11 PM | adfrw (0644) |
| LICENSE | 3,145 | File | 3/23/2010 2:29:11 PM | adfrw (0644) |
| synapp2.release_notes.txt | 3,664 | TXT File | 3/23/2010 2:29:11 PM | adfrw (0644) |

3 files and 3 directories. Total size: 8,674 bytes

A view of the SynApp2 installation directory from FileZilla

The GUI for the SynApp2 web application generator is itself, a SynApp2 powered application and is supported by the MVC framework in exactly the same way as generated applications. This speaks to the versatility of the framework. You can do just about anything with it.

Below you can see an entire application as generated by SynApp2 according to the steps detailed in the [SynApp2 Walk through No. 1](#) document:

| Remote site: /httpdocs/synapp2/census | | | | |
|---------------------------------------|----------|-------------|---------------------|--------------|
| Filename / | Filesize | Filetype | Last modified | Permissions |
| .. | | | | |
| city.htm | 11,891 | HTML Doc... | 4/9/2010 8:42:28 AM | adfrw (0644) |
| city.report.htm | 3,316 | HTML Doc... | 4/9/2010 8:42:28 AM | adfrw (0644) |
| custom.inc.php | 688 | PHP File | 4/9/2010 8:42:29 AM | adfrw (0644) |
| index.html | 821 | HTML Doc... | 4/9/2010 8:42:29 AM | adfrw (0644) |
| synapp2.inc.php | 1,629 | PHP File | 4/9/2010 8:42:29 AM | adfrw (0644) |
| temp_conv.htm | 1,377 | HTML Doc... | 4/9/2010 8:42:29 AM | adfrw (0644) |
| welcome.htm | 1,636 | HTML Doc... | 4/9/2010 8:42:29 AM | adfrw (0644) |
| 7 files. Total size: 21,358 bytes | | | | |

The census *application sub-directory* show the SynApp2 generated app files, plus a custom page - temp_conv.htm

With appropriate directory/file permissions you can generate applications on any server – local or remote. If you don't want that capability to be available, remove the synapp2 application sub-directory from the installation directory on the server. If you change your mind, at some point, you can put it back. And, if you want the capability to be available, you can control access with authorization entries in synapp2/synapp2/custom.inc.php.

Configuration

Configuration of SynApp2 is simple and flexible. Two files in a separate sub-directory control everything. If you follow a few simple naming conventions, you can create a single, universal configuration that can be maintained under source code control that works on all of servers you deal with. You don't have to change anything unless you add a new server/domain. In the mean time you just copy your apps to your servers. The details are all taken care of. It just doesn't get any easier.

| Remote site: /httpdocs/synapp2/_config_ | | | | |
|---|----------|-------------|----------------------|--------------|
| Filename / | Filesize | Filetype | Last modified | Permissions |
| .. | | | | |
| access.inc.php | 2,167 | PHP File | 7/15/2009 1:13:04 PM | adfrw (0644) |
| engine.inc.php | 1,442 | PHP File | 8/31/2009 9:02:57 AM | adfrw (0644) |
| index.html | 1,849 | HTML Doc... | 8/31/2009 9:02:57 AM | adfrw (0644) |
| synapp2.install.MySQL.txt | 6,578 | TXT File | 8/31/2009 9:02:58 AM | adfrw (0644) |
| synapp2.install.Oracle.txt | 2,934 | TXT File | 8/31/2009 9:02:58 AM | adfrw (0644) |
| synapp2.users.MySQL.txt | 629 | TXT File | 8/31/2009 9:02:57 AM | adfrw (0644) |
| 6 files. Total size: 15,599 bytes | | | | |

The _config_ sub-directory and files

The key feature of the universal access configuration and control mechanism is the ability to map **database** and **admin-user** *name prefixes* for different servers/domains. This feature

allows you to work with generic – logical names – within your apps. When your apps are deployed, the generic/logical names are automatically mapped to physical names. You just copy you app files up to the server and you're done. This is super convenient when you deploy on servers/domains where you can't always establish a database or create an admin-user with just any old names you want.

The details are in the `synapp2.install.XXXX.txt` files, but the essence of the idea follows.

Logical application database name: `projtrk`

Logical application database admin-username: `projtrk`

Logical synapp2 admin-database: `synapp2`

Logical synapp2 admin-username: `synapp2`

For server/domain `localhost` use name prefix:

For server/domain `www.ccxyzzzy.com` use name prefix: `plugh_`

On `localhost` use database and user names: `projtrk`, `synapp2`

On `www.ccxyzzzy.com` use database and user names: `plugh_projtrk`, `plugh_synapp2`

Authentication and Authorization

There are four user authentication methods available: `synapp2`, `app`, `direct`, and `enterprise`. The default mechanism for MySQL is `synapp2`. Oracle uses `direct`.

Synapp2 Method

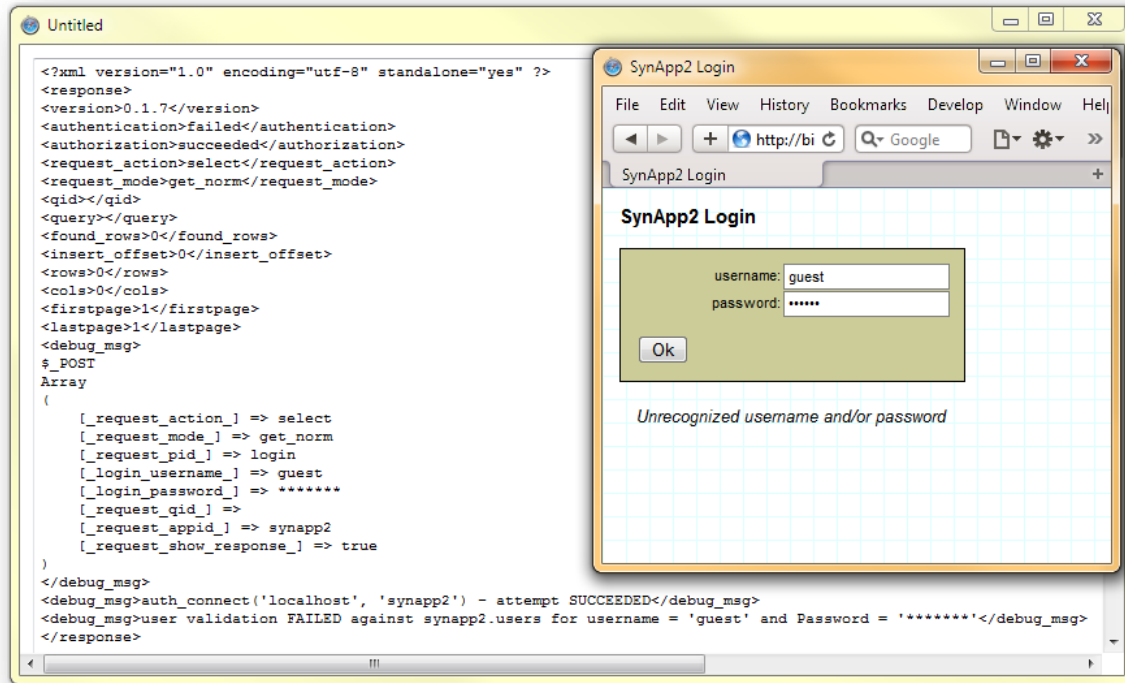
In a nutshell, `synapp2` authentication method uses a dedicated database and admin-username to store usernames and passwords. If you successfully login against that, then you're *authenticated*. Once you're authenticated, SynApp2 can see if you're *authorized* to access the application database. That involves the application `custom.inc.php` file. If you pass that test, then SynApp2 will connect you to the application database using a mapped admin-username (not usually the one you logged in with).

By default, on a (local) server (when no prefix is defined) the application database(s) are all accessed with the (privileges of) the `synapp2` user. One of the installation steps is to create the `synapp2` user in MySQL.

On a public/remote server (where there is a prefix defined) the application database is accessed with the (privileges of) the `<prefix><database_name>` user. This all works if you create both a database and a user with the same prefixed name. That's the convention. It's possible to change that if you need to. Similarly, on the public/remote server you're going to need a `<prefix>synapp2` database and `<prefix>synapp2` user. This is where you put the usernames and passwords for *that server*.

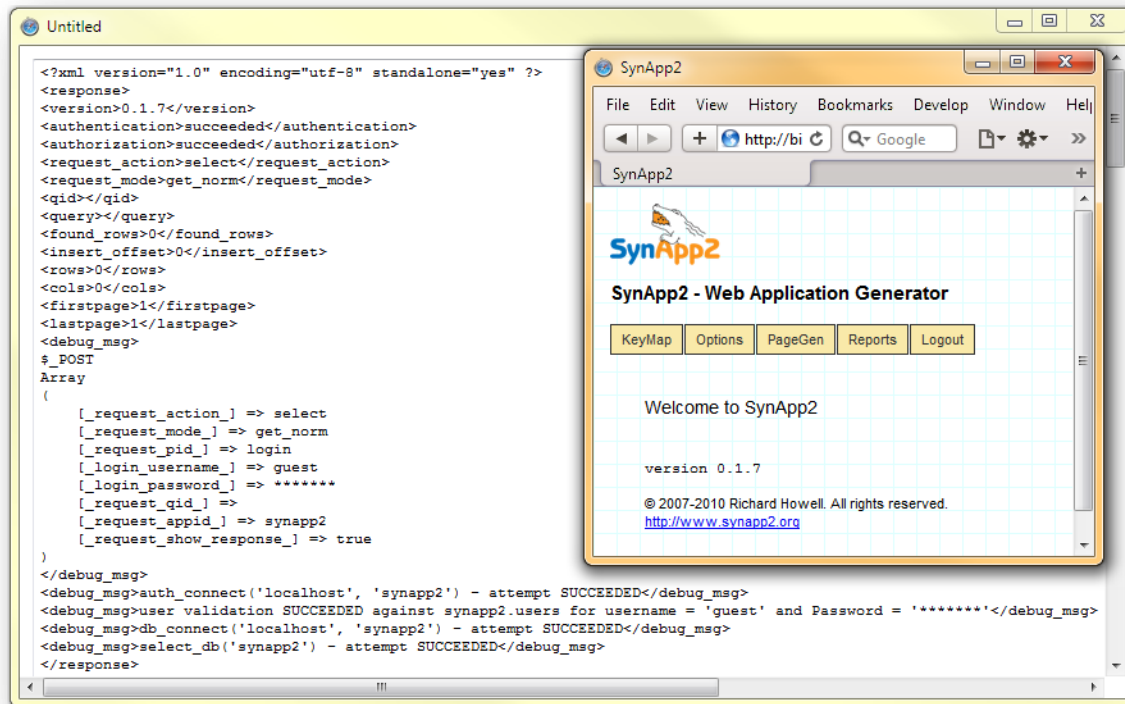
In the next two panels, look at the diagnostics at the bottom of each message window.

First login attempt (failure):



The exchange response for an unsuccessful authentication/user validation

Second login attempt (success):



The exchange response for a successful authentication and authorization

In both of the preceding login attempts, the connection to the `synapp2` authentication database was successful. But, the first attempt failed because the user name and password combination couldn't be reconciled.

There are a variety of *other* ways the authentication and authorization can fail to succeed. Use the debug message window and diagnostic messages to systematically work through what does and doesn't work. It's all pretty straightforward stuff. Look in `_shared_/access.php` for the sequence of events.

SynApp2 uses a session cookie on the server to stash a token when the login is successful. Make sure you do whatever it takes to enable session cookies on your server(s).

Direct Method

There are several other authentication mechanisms. We'll just touch on the one used, by default, for Oracle [*10g Express Edition*]. With the `direct` method, there is no need for, nor is the `synapp2` authentication database used.

Oracle handles the authentication directly. Both the developer and the user(s) log into SynApp2 and/or the generated application with the username and password of the schema owner. In the real world, that's not such a great idea, so you'll probably want to delve into creating application users and granting roles and so forth. Check out the following link for ideas: <http://www.oracle-base.com/articles/misc/SchemaOwnersAndApplicationUsers.php>

To use Oracle [*10g Express Edition*] as the database engine, you'll need to uncomment one line in `_config_/engine.inc.php`, so it looks like the following:

```
$engine = ENGINE_OCI;
```

Remove the double slashes (in front of the statement) and save the file.

An installation instance of SynApp2 can only use the configured database engine. So, if you want to use SynApp2 with more than one database engine, on the same server, you'll need to have an installation instance for each engine. That's easy, just make a different application directory and put the distribution files there with appropriate configuration and it's good to go.

App and Enterprise Methods

The remaining authentication methods provide flexibility in two areas. The `app` method relies on a user authentication table of user names and passwords in the application database. The `enterprise` method is completely open. You can extend the access class with your own PHP code to interface with an enterprise level authentication system of any size, shape or form. How you do it is up to you. Upon success, just set the 'happy bits' in the access class data members appropriately and SynApp2 should go merrily about its business.