

# **Central Christian Church Check-In Wizard**

## **Developer's Guide**

Version: 1.2.0 (rev 12)  
Last updated: 10/26/2009

Maintained by  
Nick Airdo ([nick.airdo@cccev.com](mailto:nick.airdo@cccev.com))  
Jason Offutt ([jason.offutt@cccev.com](mailto:jason.offutt@cccev.com))

**Arena Community Documentation**  
**Arena 2009.1**



## Revision History

Version	Date	Editor(s)	Description
1.00	2/10/2009	Jason Offutt	Initial version
1.01	10/26/2009	Jason Offutt	Updating notes to v1.2.0

## Table of Contents

<b>CHECK-IN UNDER THE HOOD.....</b>	<b>4</b>
1.00: Commentary on Architecture .....	4
1.01: UI State Overview .....	5
1.02: Occurrence Type Attributes .....	11
<b>DATABASE SETUP .....</b>	<b>13</b>
2.00: Tables.....	13
2.01: Stored Procedures .....	13
<b>APPLICATION SETUP .....</b>	<b>16</b>
3.00: Overview of Providers .....	16
3.01: Security Code Provider.....	16
3.02: Printer Label Provider .....	17
<b>SKINNING THE CHECK-IN WIZARD .....</b>	<b>19</b>
4.00: CSS Overview .....	19
4.01: Suggested Best Practices.....	20
4.02: Templating .....	20

## Check-In Under the Hood

### 1.00: Commentary on Architecture

While in our planning stages of this project, Nick and I made the decision to take this module away from a more traditional multi-page format that passes variables between pages via query string or session variables. In these early stages, we decided we wanted it to feel more responsive: less like a web application and more like a desktop app.

Our solution to this issue was to make the entire module a single page (ascx), rather than several. This meant jamming more into a single markup and code-behind file set, but it allowed us great flexibility. Architecting our project this way allowed us to create our own hacked version of a lazy-loaded finite state machine... in a stateless environment.

On the server side the Check-In Wizard is, in itself, a single-page module that consists of a bit of state logic to show and hide different panels (or views) from the browser. We wrapped all these panels inside of an Update Panel to give it that more responsive AJAX feel.

On the client side, the Check-In Wizard has a bit of built-in smarts as well. We've implemented a JavaScript state recognition system that allows the client to be state aware too. We use this state knowledge on the client to manage auto-refreshing or auto-submitting the form dependent on the current state. We've also moved a fair amount of the client interaction elements, such as family member selection, to the client for usability purposes.

Nick and I didn't want our module to be too smart, or have too much going on. After all, it's already got enough going on with state management and setting UI rules. We approached the problem with a traditional MVC design pattern in mind and made a very conscious effort to make the UI as "dumb" as we possibly could with respect to the inner workings of the Arena. We encapsulated all of the business logic in the [Controller](#) class. [Controller](#) acts as a façade to much of the Arena Framework. Any data retrieval, data filtration and enforcement of business rules are handled by the [Controller](#).

Additionally, we realized that there are a few aspects of Check-In that churches would want to do differently. We've implemented Arena's convention of the Provider pattern for two pieces of the application. Security Codes and Name Tags/Badges are fully customizable. Currently, we're including Central's version of Security Codes and Name Tags. More detailed description of those pieces can be found in Section 3.00.

Central Christian Church's decision to build this module hinged on three major aspects of the check-in process that Arena's version did not support. Our Children's Ministry needed to place children into classrooms not only based on age or grade, but also by ability level, special needs and last name. To solve this problem, we created the [OccurrenceTypeAttribute](#) class. I'll get more into the [OccurrenceTypeAttribute](#) class later in this section.

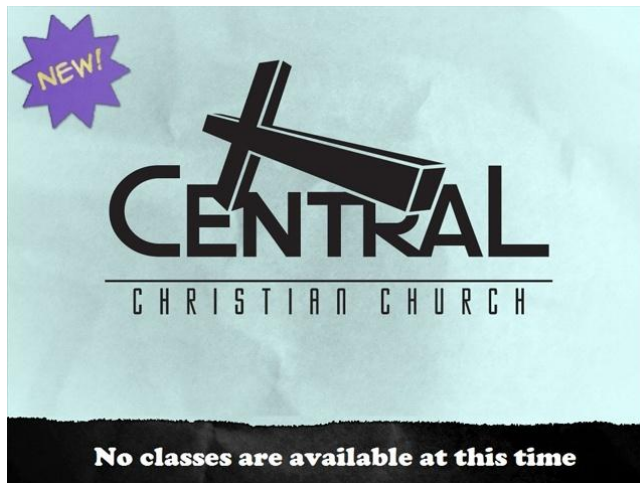
## 1.01: UI State Overview

A big part of Central's Check-In Wizard is its state management feature. All of the UI's flow is dictated by this state logic from the Wizard's code-behind file (CheckInWizard.ascx.cs). This state awareness trickles down to the client via a JavaScript that looks at what <div> tag is being rendered to the DOM, and applying client-side logic accordingly.

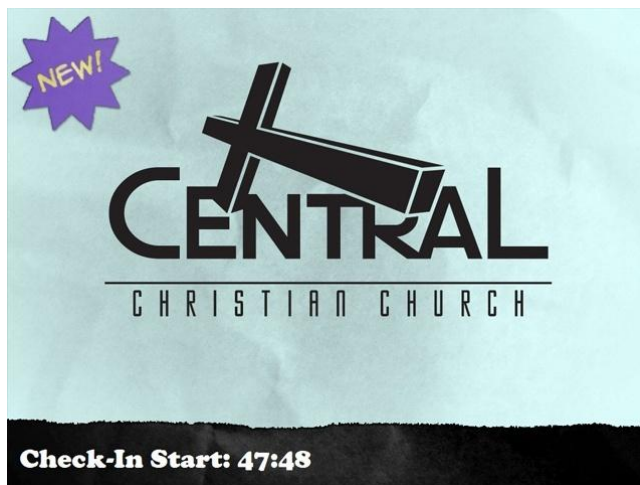
The Check-In Wizard has nine different states. In this section, I'll be giving a brief overview of each state, and what is happening behind the scenes.

- 1. Init State:** This is the Wizard's default state. The check-in process begins here. Init State is a little unique in the sense that it has three "mini-states". Depending on whether or not there is an *Occurrence* currently going on, if there is an *Occurrence* in the future that is not active yet, or there is no *Occurrence* detected at all, Init State will be rendered differently:

- a. **No Occurrence Detected:**



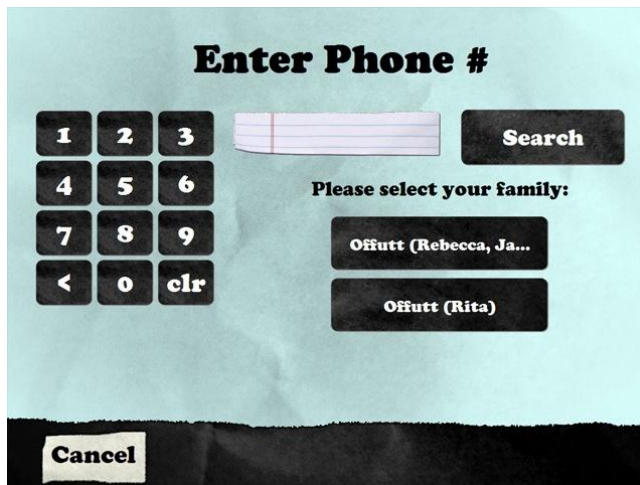
- b. **Future Occurrence Detected:**



c. **Active Occurrence Found:**



- 2. Family Search State:** When a user clicks the *Search By Phone* button from the Init State, the Wizard changes the view to display the Family Search State. Family Search is fairly strait forward. The phone keypad is all handled by JavaScript. Initially, we had some issues with the speed of the keypad when clicking the same button repeatedly. To address that issue, we ended up going into the registry of each Kiosk and setting the double click speed to 50 ms. Once the minimum search length has been reached (default is 10 characters, definable via module setting), or the search button is pressed, a DataList will appear below the search box displaying a list of families.



- 3. Select Family Member State:** When we arrive on this view from either scanning a bar code or picking a family after searching by phone, the `Controller` class is searching the database and grabbing all of the people who are eligible to check in based on `OccurrenceType`. Related people are included if any relationships are defined in the *Relationship Type List* module setting. As it binds the collection of people into the `DataList`, if it finds any people who have already checked in (by searching for an `OccurrenceAttendance` record), that person's button will be disabled in the `DataList`.

After the `DataList` has been bound, all of the interaction is handled on the client. We've implemented a solution using jQuery to handle clicking on each person. As you select or de-select people on the list, a hidden field's value is populated with a comma-delimited list of `PersonID`'s. When the *Next* button is clicked, that hidden field's value is split into an array and stored in session.

The screenshot shows a mobile application interface for the 'Offutt Family'. At the top, it says 'Hello, Offutt Family!' and 'Select All Children Attending Today'. Below this, there are six buttons arranged in two columns. The left column contains 'Lasty', 'Crutchy', and 'Walker', each with an unchecked checkbox. The right column contains 'Crawley', 'Infan', and 'Mugsy', each with a checked checkbox. At the bottom, there are two buttons: 'Cancel' and 'Next'.

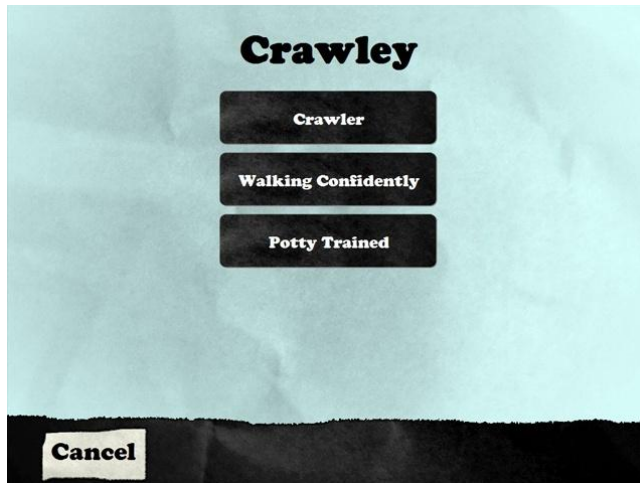
- 4. No Eligible People State:** This state will only be shown when a family is selected that does not have any relatives who are eligible to check in.

The screenshot shows a mobile application interface with a light blue background. At the top, it says 'No eligible people found' in red text. At the bottom, there is a 'Cancel' button.



**5. Select Ability Level State:** This view's visibility is entirely optional based on configuration via the *Require Attendee Abilities* module setting. When it is shown, it will recurse through each child stored in session and show a DataList of ability levels. A couple important things to keep in mind:

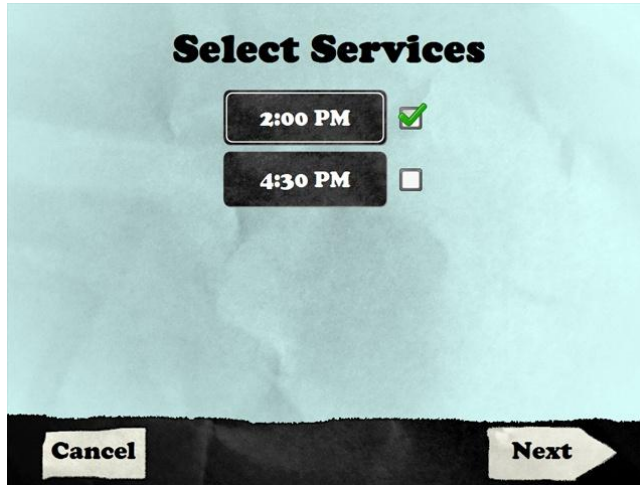
- a. If you have defined the *Max Ability Age* module setting or left it at its default of 3.5 yrs old, any child who's attained this age or higher and has not had their ability level set to the maximum will have it set for them.
- b. This view will not allow a child to regress in ability level. Ability levels that have already been attained won't be shown. The most recent ability level attained will be displayed at the top of the list.



**6. Select Service Time State:** The Service Time view allows the user to select one or more services to attend. Unlike Select Family Members, Select Service Times is primarily handled on the server, as there are a couple specific rules it needs to follow. This was much easier to enforce on the server, rather than with JavaScript.

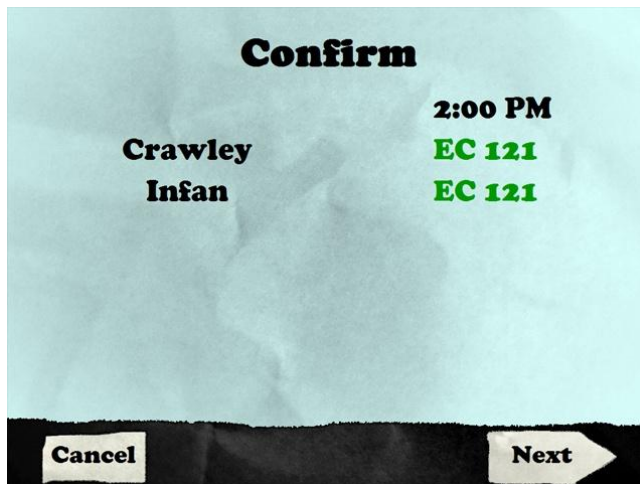


Consecutive service times may be selected, but the earliest service time must be selected first. As the user selects services in the list, the server determines which buttons on the list to enable/disable based on the user's actions.



The 'Select Services' screen features a light blue background with a dark blue header bar. The title 'Select Services' is centered in bold black text. Below the title, there are two service time options: '2:00 PM' and '4:30 PM', each in a dark blue rounded rectangle. To the right of '2:00 PM' is a green checkmark icon, and to the right of '4:30 PM' is an empty square checkbox. At the bottom, there is a dark blue bar with two white buttons: 'Cancel' on the left and 'Next' on the right, which is a right-pointing arrow.

- 7. Confirmation State:** The Confirmation state dynamically builds a table of all attendees checking in and displays a column for each service time they're checking into. The first column will display the attendee's Arena nick name and each subsequent column will display the name of the room (or [Location](#)) they are going to be sent to.



The 'Confirm' screen has a light blue background with a dark blue header bar. The title 'Confirm' is centered in bold black text. Below the title, there is a table with two columns. The first column contains the names 'Crawley' and 'Infan' in bold black text. The second column contains the service times '2:00 PM', 'EC 121', and 'EC 121' in bold green text. At the bottom, there is a dark blue bar with two white buttons: 'Cancel' on the left and 'Next' on the right, which is a right-pointing arrow.

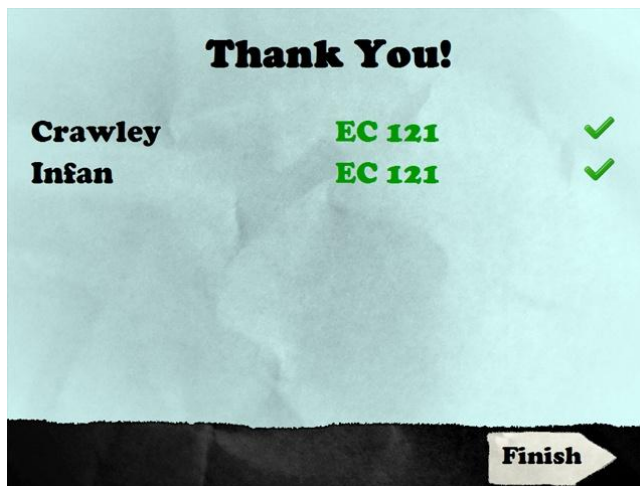
- 8. Results State:** The final state in the Wizard's flow is the Results view. On the surface, this view appears to only be building another dynamic HTML table. In the background, however, it is issuing a request to the [Controller](#) to check each attendee in and print a label for them.

Each table row that is generated is output from the [Controller](#), and will change dependent on whether the child was successfully checked in and whether their badge printed successfully. The table row will look similar to the Confirm View's table. It will display the attendee's Arena nick name, the first [Location](#) name they are being sent to, and a status message indicating success, check-in failure or print failure.

Check-in failure should only ever happen if a SQL exception is generated while attempting to insert a new [OccurrenceAttendance](#) record into the database.

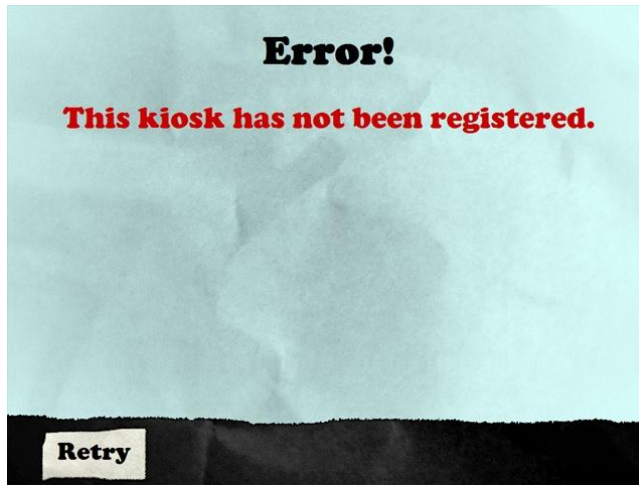
Print failures can happen for a couple reasons. The most common would be that the printer that the nametag is being sent to has not been defined properly in Arena; possibly an incorrect network path, IP address or is absent from Arena all together. Another, more obscure reason, might be that the print service on the server has crashed and resetting the application pool in IIS may be needed.

The second issue should be very rare. I include it here because we ran into this problem while trying to send large amount of print traffic to the server running in a Hyper-V virtualized Windows Server 2003 R1 environment (upgrading to R2 corrected the problem). There is very little documentation on the web regarding this issue.



- 9. Invalid Kiosk State:** If a system has not been registered as a Kiosk in Arena, you will see this state. It should be very rare, and will only occur if there is an issue with your Kiosk setup in the Arena database.

Clicking the *Retry* button attempts to cycle the Wizard module back to the Init State.



## 1.02: Occurrence Type Attributes

Central Christian Church's Check-In Wizard allows the ministry consuming the application to define criteria linked to [OccurrenceTypes](#). They are represented in the Wizard's business logic through the [OccurrenceTypeAttribute](#) entity class.

[OccurrenceTypeAttribute](#) relates to [OccurrenceType](#) by holding a reference of the corresponding [OccurrenceTypeID](#). An [OccurrenceType](#) can have multiple Ability Levels tied to it through the implementation of Arena's [Skeleton/Bone](#) data structure.

[OccurrenceTypeAttribute](#) in its current form has a few publicly exposed properties:

- [IsSpecialNeeds](#) – A Boolean value that indicates whether or not the [OccurrenceType](#) supports Special Needs attendees.
- [LastNameStartingLetter](#) – The starting alphabetical character that will be supported by the corresponding [OccurrenceType](#).
- [LastNameEndingLetter](#) – The ending alphabetical character that will be supported by the corresponding [OccurrenceType](#).
- [AbilityLevelLookupTypeID](#)s – This property allows us to tie multiple Ability Levels to the [OccurrenceType](#). It represents a [List<int>](#) of [LookupType](#) ID's.

An `OccurrenceTypeAttribute` is not required for every `OccurrenceType` defined in your Arena install. If you delete an `OccurrenceTypeAttribute`'s matching `OccurrenceType`, the database will do a cascading delete on the `OccurrenceTypeAttribute`.

At the time of this guide's writing, the best practice would be to create an `OccurrenceType` for each service or class. If that particular `OccurrenceType` requires any special criteria, the administrator will be able to add any of the above listed criteria to enable the Check-In Wizard to filter through. As this is the *Developer's Guide*, I'll skip the gory details here. You can find a more thorough explanation in the *Administrator's Guide*.

Additionally, if and when Arena implements the criteria listed above into the `OccurrenceType` class in the Arena Core Framework, we will remove these properties and database column from our `OccurrenceTypeAttribute` class and table and make use of the default Arena implementations.

## Database Setup

### 2.00: Tables

When installing the Check-In Wizard, three database tables will be created by default:

- `cust_cccev_ckin_occurrence_type_attribute`
- `cust_cccev_ckin_security_code`
- `cust_cccev_applog`

The `cust_cccev_ckin_occurrence_type_attribute` table is the database representation of Central's `OccurrenceTypeAttribute` custom entity object. There's not much to the table beyond what's been mentioned above in the class outline for `OccurrenceTypeAttribute`.

The `cust_cccev_ckin_security_code` table is used by Central's provider for creating security codes. We've included a SQL job that will write null values to the table each week so it can be reused later. This table holds only two columns: `assigned_date` and `unique_key`. It does not hold the actual security code generated to be associated with the attendance record. I'll get more into how this database functionality works when I discuss stored procedures.

Additionally, this table is created by default by the SQL installer scripts included with the module. However, it is not mandatory to have for the Check-In Wizard to work. We've implemented the management of security codes as a provider. If you plan on writing your own custom provider, or already have a provider in place, the `cust_cccev_ckin_security_code` table can be safely deleted from your database.

The `cust_cccev_applog` table is used by the Check-In system to log possible matches when filtering the list of available `Occurrences` to check the attendee into. If enabled, the system will attempt to write the steps it takes through the filtration algorithm to the database. This feature is configurable through a `LookupType` called "Cccev Application Log Type". Setting the Check-In Lookup's "IsEnabled" value to "false" will disable it. This feature is disabled by default on installation.

### 2.01: Stored Procedures

In addition to the two tables, the Check-In Wizard's SQL installer script will also create several stored procedures in your database.

- `cust_cccev_ckin_sp_del_occurrenceTypeAttribute`
- `cust_cccev_ckin_sp_get_occurrence_attendance_byPersonIDAndStartDate`
- `cust_cccev_ckin_sp_get_occurrenceByCategoryAndDate`
- `cust_cccev_ckin_sp_get_occurrencesBySystemIDAndDateRange`
- `cust_cccev_ckin_sp_get_occurrenceTypeAttributeByGroupId`
- `cust_cccev_ckin_sp_get_occurrenceTypeAttributeByID`
- `cust_cccev_ckin_sp_get_occurrenceTypeAttributeByOccurrenceTypeId`

- cust\_cccev\_ckin\_sp\_get\_personWithHealthOrLegalNotesByAgeOrGrade
- cust\_cccev\_ckin\_sp\_get\_security\_code
- cust\_cccev\_ckin\_sp\_insert\_security\_code\_addNumbers
- cust\_cccev\_ckin\_sp\_save\_occurrenceTypeAttribute
- cust\_cccev\_ckin\_sp\_update\_security\_code\_clearAssignDate
- cust\_cccev\_ckin\_sp\_get\_location\_head\_count\_by\_date

The majority of these are your standard CRUD operations. There are, however, a few exceptions that I'll go over in a bit more detail.

The `cust_cccev_ckin_sp_get_occurrence_attendance_byPersonIDAndStartDate` procedure is used in the Init and Phone Search states of the Wizard to check whether or not a person has checked in. This allows us to preemptively check to see whether or not a given person has signed in before proceeding with the check-in process. This procedure becomes significantly important when auto-advancing through states.

`cust_cccev_ckin_sp_occurrencesBySystemIDAndDateRange` is used in the Init State. This procedure allows us to look ahead into the future to see if there are any upcoming services to open check-in for. The procedure also allows us to constrain the available occurrences based on the location of the kiosk. A kiosk running Central's Check-In Wizard can only check in to events that are tied to the same location the kiosk is tied to.

`cust_cccev_ckin_sp_get_security_code` is used by Central's default Security Code provider for creating a security code for a child's name tag and claim ticket set. This proc will write a record to the `cust_cccev_ckin_security_code` table's next non-null row. The proc returns a `varchar` composed of two randomly chosen letters and the row's index expressed as a 4-digit number. This security code is printed on both the name tag and the claim card. Central's Servant Ministers will not release a child to a parent without the matching claim ticket.

`cust_cccev_ckin_sp_insert_security_code_addNumbers` and `cust_cccev_ckin_sp_update_security_code_clearAssignDate` are unique in the sense that there isn't any working code within the Check-In Wizard that depends on these two procs. They're really only used in the database itself.

`cust_cccev_ckin_sp_insert_security_code_addNumbers` is only used in the installer script to populate the `cust_cccev_ckin_security_code` table (used by our default Security Code provider) once it's been created. This proc will create 9999 rows (skipping 911 and 666), but can be altered by passing a different value to its `@endVal` parameter.

`cust_cccev_ckin_sp_update_security_code_clearAssignDate` is sort of a cousin to `cust_cccev_ckin_sp_insert_security_code_addNumbers`. Once the table has been created, `cust_cccev_ckin_sp_insert_security_code_addNumbers` can be run to clear out the values in each record and reset them back to null. By default, we've got a SQL job that's scheduled to run every Friday to clear the security code records out before the weekend.

`cust_cccev_ckin_sp_get_location_head_count_by_date` allows the system to get attendance head counts of `Occurrences` at a specific time. We basically took the existing Arena stored procedure and added a start time parameter to it, rather than using the current date and time. This procedure was essential to enable our system to do room balancing.



## Application Setup

### 3.00: Overview of Providers

During our development process, we took measures to make the Check-In Wizard as flexible as we could make it. Certain pieces of functionality had to be able to be made specific to each church or ministry using the software. With that realization, we decided to apply the Provider Pattern in two specific areas. The core functionality of the Wizard module can work anywhere, but we identified two specific areas that other churches might want to do differently than Central.

The first area is creating a custom security code for name tags and claim tickets that get printed out. Central's approach to this was to create a six character (two letters and four numbers) security code. Realizing that every church might have their own way of doing this, we decided to encapsulate that part of the Wizard's functionality. We've included our implementation as an optional way to offer a complete package. If your ministry requires a different set of security codes, or has a different way of creating them, then the provider interface is there for you to extend and create your own.

The second facet of this project that we decided to offer more flexibility to is the physical name badge and claim ticket themselves. The entire label is customizable, along with the printing of that label. Again, we've included our own implementation to provide a complete working package upon delivery of our module, but if your ministry has a very specific need, the provider interface is available to be extended upon.

#### 3.01: Security Code Provider

Our implementation of the Security Code Provider begins by extending an interface we created. The interface `ISecurityCode` allows a developer to create their own custom implementation of the Wizard's security code functionality. `ISecurityCode` defines a single method called `GetSecurityCode()`, which returns a string.

In order to tie your custom provider class to the Check-In Wizard, there are two very important steps you must take in order to register it with Arena:

The Check-In Wizard's SQL installer script should have created a new `LookupType` in the Arena Administration section named "Check-In Security Code System". This `LookupType`, has defined Qualifier 2 as "Namespace" and Qualifier 8 as "Class".

You'll need to create a new `Lookup` in the Check-In Security Code System `LookupType` alongside the default one. Set the Namespace qualifier to point to the assembly your custom provider class resides in (i.e.: `Arena.Custom.Cccev.CheckIn`). Set the Class qualifier to the fully qualified path of your custom implementation (i.e.: `Arena.Custom.Cccev.CheckIn.Entity.CccevSecurityCode`).

Once the new Lookup has been saved, you'll want to copy down that new `LookupID` in the ID column of the DataGrid.

Our SQL installer script should have created a new Organization setting on your Arena install named "Cccev.SecurityCodeDefaultSystemID". You'll need to edit that setting and replace the value of the default `LookupID` with the one you just created.

With all that done, you'll want to refresh your current portal's server cache, and your new settings should be loaded in your Arena server's memory.

Central's provider makes some pretty heavy use of database functionality via the `cust_cccev_ckin_security_code` table and the stored procedures mentioned earlier: `cust_cccev_ckin_sp_insert_security_code_addNumbers`, `cust_cccev_ckin_sp_update_security_code_clearAssignDate`, and `cust_cccev_ckin_sp_get_security_code`.

As I touched on earlier, the SQL installer scripts included with the Check-In Wizard module will create the above mentioned table and stored procedures in your database along with a SQL job. `cust_cccev_ckin_sp_insert_security_code_addNumbers` will only ever be used when creating the `cust_cccev_ckin_security_code` table. By default the table will be created, and the insert procedure will fill it with 9999 null rows.

`cust_cccev_ckin_sp_get_security_code` is run every time a child checks in through the Check-In Wizard and we issue them a security code from our provider. This procedure will grab the next null row in the database and assign the current date/time and a new GUID to the record. The full security code itself is not stored in the database as a precautionary method.

Every Friday, the SQL job created by the installer script will run the `cust_cccev_ckin_sp_update_security_code_clearAssignDate` procedure against the `cust_cccev_ckin_security_code` table, writing a null value to every date and GUID value for each record so the table can be reused each week.

### 3.02: Printer Label Provider

Our second Provider implementation is for the printing of our labels (name tags, claim tickets, etc). Again, we've supplied our implementation as a default, but this aspect of the Check-In Wizard is completely extendable if your ministry has very specific requirements.

Central's approach was a little different than most. Our children's ministry, until the release of this Arena module, was using an older system that was developed in house by Nick and a few volunteers. At Central, we already had our check-in labels established and designed specifically for our children's ministry. So, our default provider class, `CccevPrintLabel` is really just a façade for our existing class, `CheckinLabel`. We've supplied both classes as part of our default implementation.

If you are interested in creating your own label functionality, we've made the `IPrintLabel` interface available to be extended. Like `ISecurityCode`, `IPrintLabel` has only one method to implement: `Print()`.

From here, the rest of the provider class configuration is identical to that of Security Code's. Our installer will write the `LookupType` for you in the Arena Administration's Lookup area. You'll want to create your own `Lookup` under the "Check-In Print Label System" `LookupType` and make sure to map your assembly name to the "Namespace" qualifier and fully qualified path to the "Class" qualifier.

The installer script should have created a corresponding Organization Setting for you as well called "Cccev.PrintLabelDefaultSystemID". You'll want to make sure to change its value to match the `LookupID` of the new `Lookup` you just created.

After a quick portal cache refresh, your new custom provider should be up and running with the Check-In Wizard module.

As of version 1.2.0, we've included a new provider (`Arena.Custom.Cccev.CheckIn.Entity.RSPrintLabel`) for Print Labels that utilizes Reporting Services. This new provider is compatible with the labels that come out of the box with Arena by passing the `OccurrenceAttendanceID` field of the `OccurrenceAttendance` record that gets generated during the Check-In process to the Reporting Services report. This provider also allows configuration for At-Kiosk or At-Location printing via `OccurrenceTypeReports`. For more information on setting up the new Reporting Services provider, please refer to the Check-In Wizard's Administrator's Guide.

## Skinning the Check-In Wizard

### 4.00: CSS Overview

The default skin that comes with the Check-In Wizard was designed with Central's specific ministry needs in mind. While you're more than welcome to use our default skin, it's more than reasonable to expect any churches who'd like to integrate our module with their Arena installation to want to tweak the look of the Wizard (or give it a complete overhaul). There are a couple things you may want to keep in mind while hacking away at our CSS (found in the *UserControls/Custom/Cccev/checkin/Misc* folder).

As I mentioned earlier, the logic that drives the flow of the various views within the Check-In Wizard is merely toggling the `Visible` property on the appropriate `Panels` within the control itself. Since each View is essentially its own complete UI, we have many CSS classes that are being reused consistently.

The footer for each page has a group of CSS classes defined that control its appearance:

- `footer`
- `footerRight`
- `footerLeft`
- `cancelButton`
- `nextButton`

These five classes essentially define the entire look and feel of the footer on just about every view.

Any buttons can be re-skinned as well. One important thing to remember: Since we've given our buttons lots of custom styling (background images, etc), we have more than one CSS class to define how they look in enabled, disabled or selected states.

- `dataButton`
- `dataButtonInactive`
- `dataButtonSelected`
- `nameButton`
- `phoneButton`

We've also defined several text-related CSS classes to control the way text is rendered out to the browser:

- footerText
- footerCaption
- footerError
- checkinText (body text and specific to <h3> tags)
- confirmText
- classroomText
- errorText
- checkinCaption
- errorCaption
- confirmError
- success
- fail

Additionally, we've implemented some styling that overrides the default AJAX "Loading" progress indicator for Arena on every state except for the Init state. If you dislike the spinning progress indicator we've added, you can simply modify the contents of `ajaxLargeProgress` CSS class to suit your taste.

## 4.01: Suggested Best Practices

When re-skinning your installation of the Check-In Wizard, there are a couple of guidelines that may make things a little easier for you.

If you are attempting to modify the default `checkin.css` file included in the module, it might be a good idea to make a backup copy of it, or start by making a copy of `checkin.css` and renaming it to something else. We have exposed a module setting that will allow you to change the CSS file associated with the page. The module setting controls the relative path of the style sheet in the `<link>` tag being rendered to the HTML page header. This is not a required module setting, and if left blank, it will point to the default file: `checkin.css`.

## 4.02: Templating

For the Check-In Wizard, we've included a blank template file that you'll need to be sure to add. The CSS and `.ascx` file define the look of the module. The module could easily be wrapped in a different template; however, I would probably recommend modifying the CSS file that controls the template's appearance to more easily accommodate a wrapped interface. The default skin/template is formatted to fill a 1024 x 768 monitor.