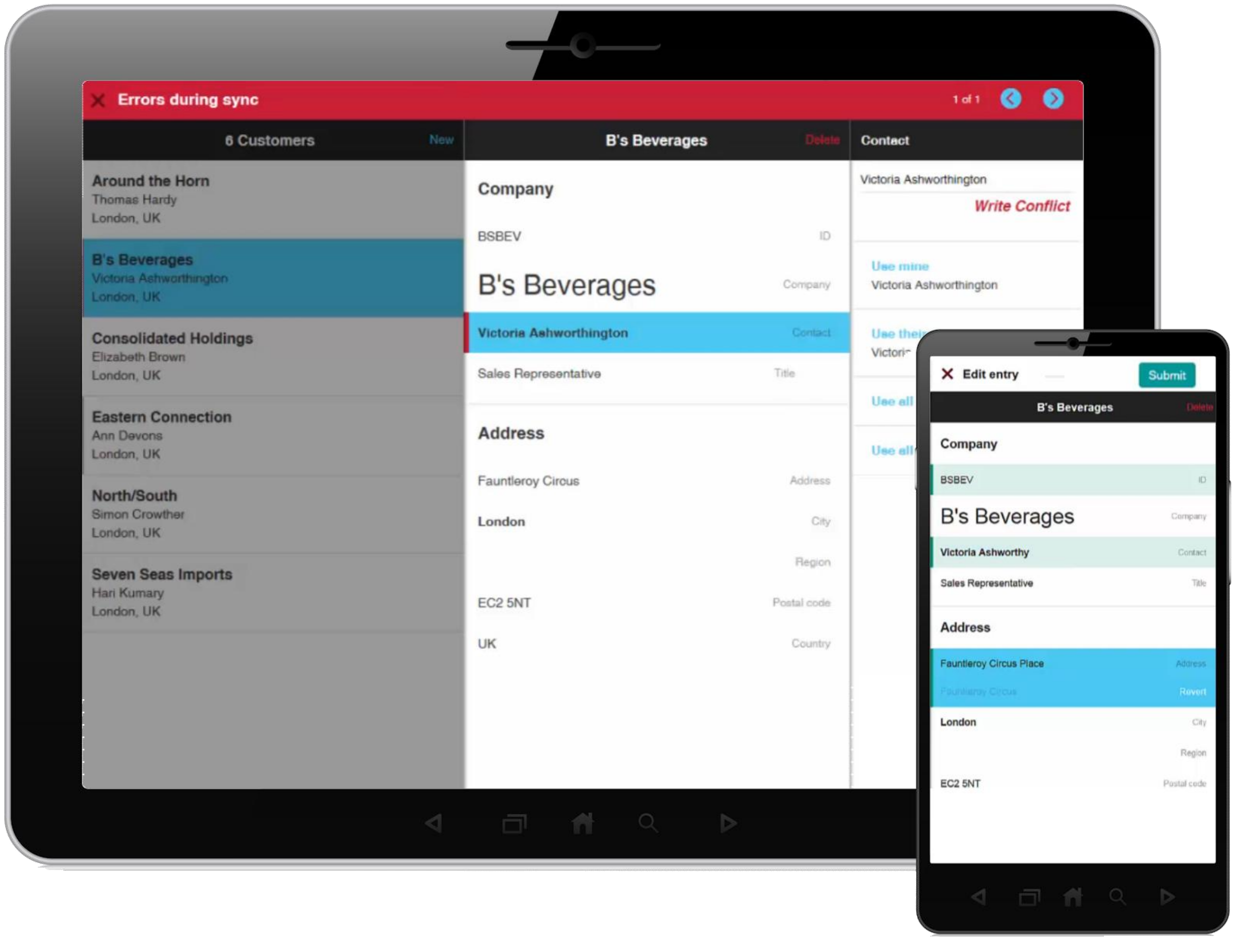
How Alpha Anywhere Supports Offline Mobile Apps

An Explanation of the Offline Features in Alpha Anywhere (Applies to Disconnected Laptops, Too)



Introduction

In my essay "Dealing with Disconnected Operation in a Mobile Business Application: Issues and Techniques for Supporting Offline Usage", I list some of the **major issues** relating to supporting offline usage. Those issues are:

- **Persisting local data** including unsynchronized transactions and app state
- Getting mobile **transactions back** to the corporate database
- **Resolving conflicts** that arise from a delayed update of a disconnected database
- Managing with the **peculiarities of mobile wireless** connectivity
- Making all this **clear to the user** where needed

The latest version of Alpha Anywhere released by Alpha Software Corporation in September 2014 (Alpha Anywhere V3) added a number of new features specifically designed to support the implementation of support for offline operation on mobile devices. The purpose of this essay is to explain the ways in which Alpha Anywhere now addresses those issues.

What is Alpha Anywhere?

Alpha Anywhere is a **model-based "low-code"** development system. That is, it provides a **visual, non-line-by-line-code interface** for specifying an application (e.g., dialog boxes, tree controls, drag-positioning, etc.). It is for **both front-end and back-end app creation**, and, very importantly, it has **programmer-friendly facilities for integrating custom code** written in common, familiar computer languages, like JavaScript, CSS, SQL, and Basic. It is not a simplistic system, but rather it is a complete, prototype-to-production environment for rapidly developing and deploying enterprise-level, cross-platform mobile and web business applications using HTML5 technologies. (To watch a video explaining this, see "Alpha Anywhere Low Code Technical Overview".)

Alpha Anywhere can be used to create quite powerful, complex applications, such as those used in the real world by large corporations as part of their operation. It can be used to create sophisticated mobile user interfaces, such as shown in "AlphaRef Reader: Tablet-first design of an app for reading reference material". However, our users were finding that a large number of the mobile apps they wanted to produce needed to work when disconnected, even if infrequently. The high leverage in rapid development that they got from Alpha Anywhere was thwarted by the difficulty of writing their own custom code to support offline. They asked for us to build that support into the system, just as Alpha has already done for many user interface, data-accessing, and other hard-to-code functionality.

For Laptops, Too

Note that while much of what is written here is about "mobile apps", Alpha Anywhere also creates regular browser apps targeted to desktops and laptops, as well as apps that work on both keyboard/mouse browsers and touch devices. Most of the new functionality for offline is just as applicable to a laptop in the field as to a smartphone or tablet.

Overview of the New Offline Features in Alpha Anywhere

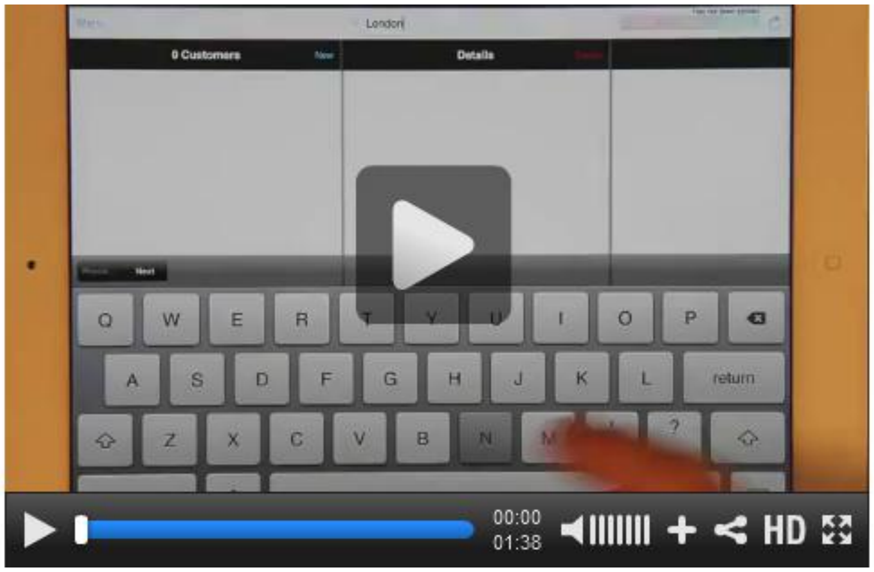
The new support for offline operation of mobile apps is extensive, with customizable functionality throughout the system. This is especially true in the **List Control**. (The List Control is one of the most common means for displaying data retrieved from a server, and has always had functionality for downloading data for display and maintaining that data in JSON form.)

Here are many of the **new features**:

- List Control support for **storing and displaying transaction details**, including changes, errors, and updates.
- List Control and server-code support for **synchronizing** stored transactions back to the server.
- List Control and server-code support for **detecting and resolving** data collision and other **errors**.
- List Control and server-code support for **incremental downloading** of updated data to the client from the server.
- List Control and server-code support for **hierarchical JSON and linked-table SQL**, including write conflict detection with roll-back.

- Granular control over multi-transaction synchronization to better deal with long synchronization operations.
- Built-in, automatic use of browser **"localStorage"** to persist data and state.
- Automatic creation of a manifest file to make use of the browser's **appCache** mechanism for **persisting web apps HTML**, CSS, etc.

Here is a short video that shows an example of using the offline support in an app created with Alpha Anywhere and running in a browser with the browser UI hidden:



A More Detailed Explanation

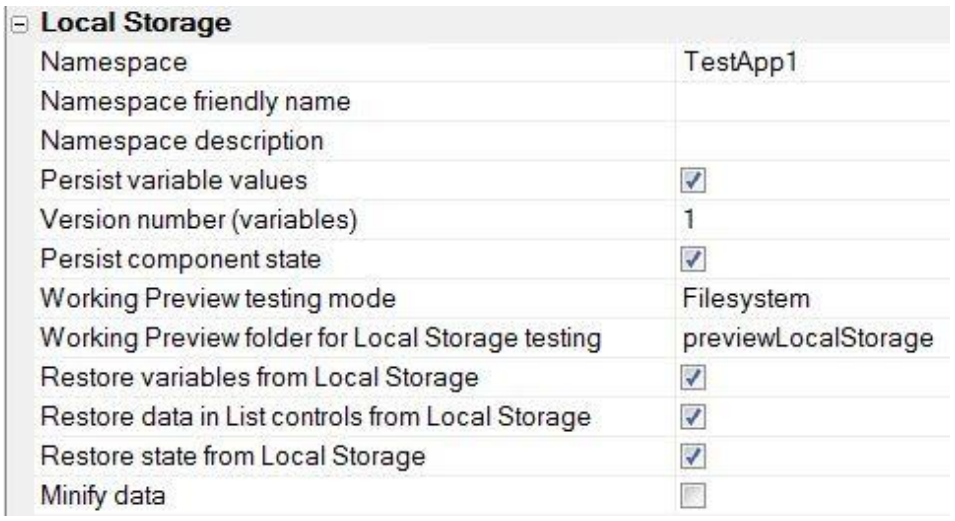
What follows is a more complete explanation of the major new features.

Persisting Local Data Including Unsynchronized Transactions and App State

Some of the issues relating to persisting are: Preserving downloaded data, preserving data updates until communicated, securing data on the device, persisting the app (for example, during phone calls or power-downs), and returning the user interface to proper state.

Alpha Anywhere now has built-in support that uses the browser's "localStorage" data persistence functionality. When building an application, there is a new set of properties, "Local Storage", in the application's settings. These let you configure the use of local storage and determine whether variable values and/or component state should be automatically persisted and restored, as well as control how the persisted data is made available during debug previewing.

Here is a screenshot of the properties:



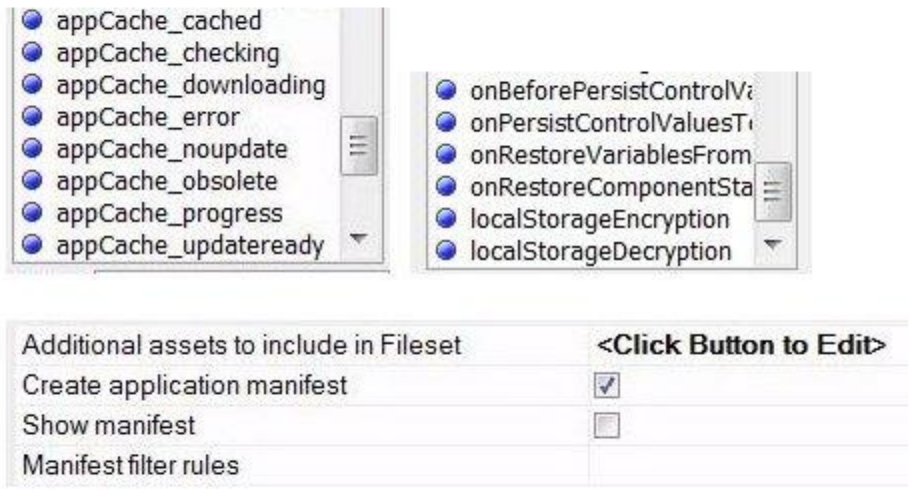
Local Storage	
Namespace	TestApp1
Namespace friendly name	
Namespace description	
Persist variable values	✓
Version number (variables)	1
Persist component state	✓
Working Preview testing mode	Filesystem
Working Preview folder for Local Storage testing	previewLocalStorage
Restore variables from Local Storage	✓
Restore data in List controls from Local Storage	✓
Restore state from Local Storage	✓
Minify data	☐

There is a button for adding local storage maintenance controls to a component. There are event handlers that can have custom code for processing data before it is saved in persistent storage as well as when it is retrieved. List controls have a "Persist data to local storage" checkbox.

Many types of controls have **automatic support for persistence** when saved and then later retrieved with these mechanisms. There are event handlers that may be used to add such support to any other controls through custom JavaScript. The saving and loading event handlers may be used to add encryption and other features to the saved data.

The "Create static HTML files" Menu item (when editing a mobile application's controls) now includes a **"Create application manifest"** and related settings, and there are new "appCache_" client-side events for interacting with the browser's standard "appcache" functionality for persisting HTML, CSS, JavaScript, and other files.

Here are some additional screenshots, including the list of appcache and local storage events:



- appCache_cached
- appCache_checking
- appCache_downloading
- appCache_error
- appCache_noupdate
- appCache_obsolete
- appCache_progress
- appCache_updateready

- onBeforePersistControlV
- onPersistControlValuesT
- onRestoreVariablesFrom
- onRestoreComponentSta
- localStorageEncryption
- localStorageDecryption

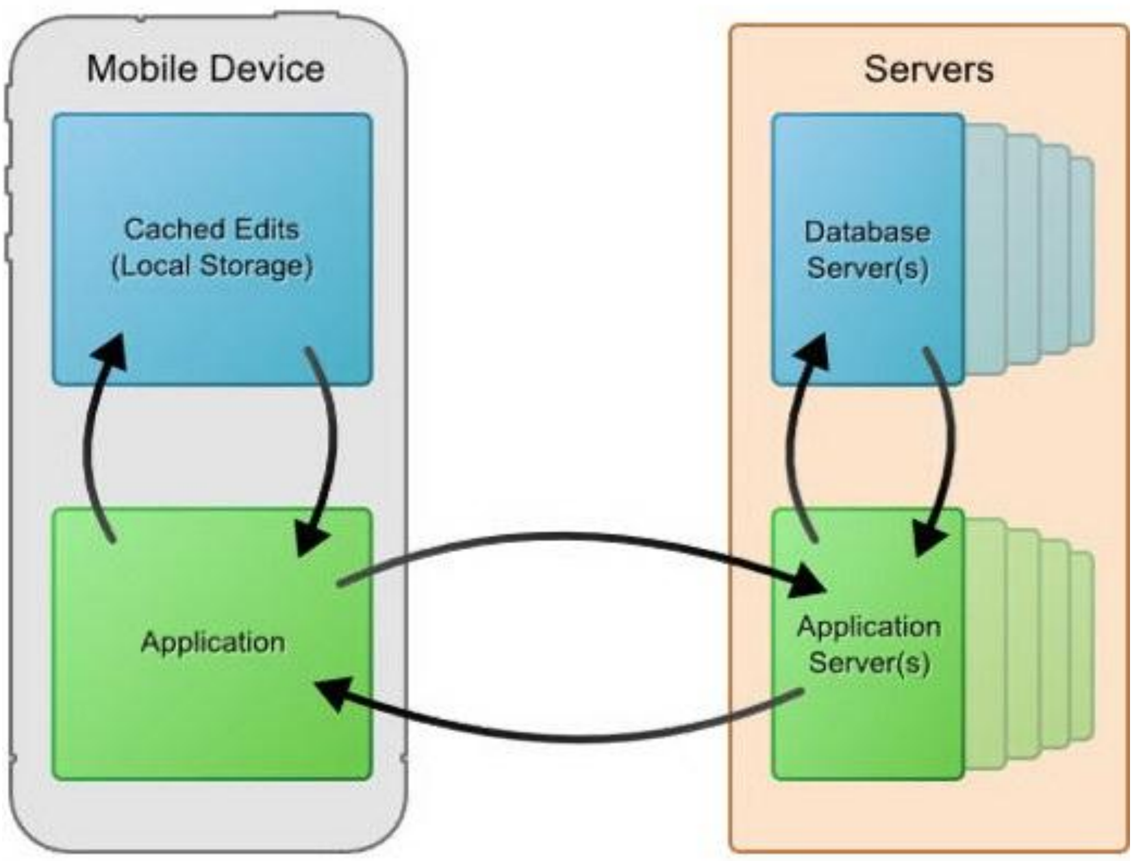
Additional assets to include in Fileset	<Click Button to Edit>
Create application manifest	☑
Show manifest	☐
Manifest filter rules	

For each of the events there is a text editor box so you can write your own custom JavaScript code.

Getting Mobile Transactions Back to the Corporate Database and Resolving Conflicts that Arise From a Delayed Update of that Database

The List Control has been extended to include built-in support for an associated "Detail View" as well as optional support for storing changes made to its data and later forwarding those changes to the server.

Alpha Anywhere uses the "store and forward", "deferred updating" style of posting transactions back to the server, as described in the other essay, as opposed to the Local Database Synchronization style. In the following diagram, you can see how the application, including the List Controls in it, saves data in local storage and then communicates directly to the server during synchronization:



The Detail View is a set of controls, such as editable text fields, that can be used to display, edit, or create data for the data items in a selected row in the list. The interaction between the Detail View and the list can **automatically keep track of any changes that the user makes** to each row.

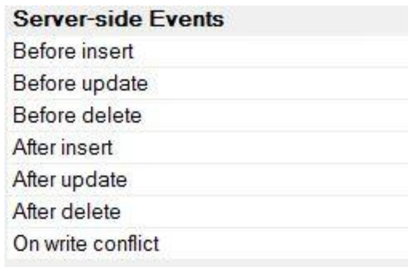
The List Control now keeps track of additional information about each row in the list, and each field in those rows. This information includes:

- Has the data been changed since it was last downloaded from the server? If so, what is the new value and what value that was last downloaded did it replace?
- Is the row a new row that has not yet been uploaded to the server?
- Is the data new data that was retrieved from the server to replace previous data from a prior download?
- Did the row have errors that were detected by the server during a synchronize operation?
- Does a field have errors that were detected by the server during a synchronize operation?

Other changes include:

- The List Control can now automatically change the CSS class of displayed list rows and detail view controls to indicate their state, such as "edited but not synchronized", "unread update row", or "error". You can see how that is used in the demo video to display rows with edits or errors.
- The synchronize code for the List Control that interacts with the server can now handle multiple stored changes, instead of just updating from the server each time new data is committed from detail controls for a single row.
- The server code can now use data included with synchronization uploads to determine if the data in the database differs from that assumed by the client (a common case with "write conflicts") or if data on the client is out of date during a "download new updates" operation.
- In the event of errors the server can either use its own algorithms for resolving those errors, or it can leave things unchanged until the client sends new instructions.
- The server can now communicate back to the client information about errors or updates.
- The error information can be used by the client to give the user control of resolving those errors. Some of the most widely used means of doing this are built-in as default code.

Here is a screenshot of some of the events that are available for customization:

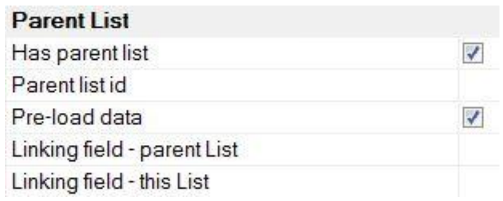


Server-side Events
Before insert
Before update
Before delete
After insert
After update
After delete
On write conflict

There is additional support related to data in JSON form:

- The List Control can now work better with hierarchical JSON data. That is, it supports lists of data that include row fields that themselves are lists. An example of this would be a list of customers, with a sub-list of orders for each customer and a sub-list of line items for each order.
- The different list controls that are used to display such data can be configured so that all of the data can be stored in one JSON-style object in the top-level list. Changes to sub-lists and their items are automatically reflected in the top-level list, including the storing of change information and the handling of synchronization errors, etc.
- The server-side code for handling synchronization can deal with such hierarchical JSON data, automatically converting it to the appropriate series of SQL commands, and deal with errors, including those that require roll-back of transactions.

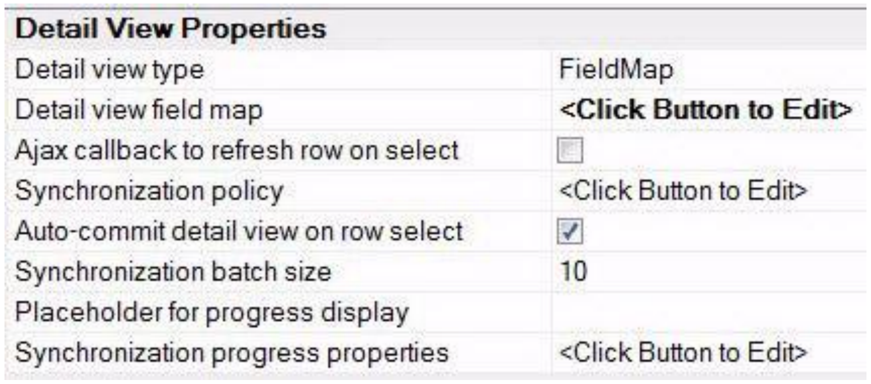
Here is a screenshot of some of the List Control settings:



Parent List	
Has parent list	✓
Parent list id	
Pre-load data	✓
Linking field - parent List	
Linking field - this List	

Managing with the Peculiarities of Mobile Wireless Connectivity

Synchronizing data in list controls can now occur in batches, with more than one row of data updated with a single server request. The size of the batches can be controlled to balance the need for speed (by minimizing the number of server requests) with the need to avoid timeouts and show helpful progress indication to the user. There is also built-in support for a progress display and an optional "Cancel" button.



Detail View Properties	
Detail view type	FieldMap
Detail view field map	**<Click Button to Edit>**
Ajax callback to refresh row on select	☐
Synchronization policy	<Click Button to Edit>
Auto-commit detail view on row select	☑
Synchronization batch size	10
Placeholder for progress display	
Synchronization progress properties	<Click Button to Edit>

Synchronization of data updates from server to client can be configured. Functionality is provided to help minimize the amount of data transferred by only downloading rows that have changed.

Data download can be used in conjunction with the local storage on the client to give the user immediate visual feedback on the list control of updates committed from a Detail View, but still send those changes back to the server for processing and perhaps other changes to be downloaded.

The system can be configured to detect changes in connectivity and change its behavior along with indications to the user.

Making All of This Clear to the User Where Needed

The main functionality added in this release of Alpha Anywhere addresses User Experience issues by providing the newly added offline support. This includes the ability to **visually indicate the state of data** being kept in the List Control and the data displayed in Detail Views. Alpha Anywhere also supports automatic updating of CSS classes to indicated modified or updated data as well as error situations. This improves the user experience by letting the user know that unsynchronized data exists on their device. These new features can also assist the user in locating updated or error-causing data.

The video shows one of the user interface styles that can take advantage of the functionality provided by this release of Alpha Anywhere. That particular implementation was created using Alpha Anywhere, with custom HTML, CSS, and JavaScript to provide the handling of edits, updates ("unread"), and errors. (The documented source of the entire app is part of the documentation for the release.) The initial configuration of the main list control, as well as many of the buttons (including the synchronize and the Submit buttons) was done by using a built-in "genie". The "genie" uses a series of dialogs to gather information about the server-side database table and desired behavior with respect to a Detail View and adds the appropriate controls to the project. In this release the controls are minimally formatted, awaiting the developer's customization. An **upcoming release** will include major new features in that area and the genie will be enhanced to support those new features to produce mobile-style applications similar to the video, complete with support for direct navigation to edits and errors.

Further Information

Further information about Alpha Anywhere is available from Alpha Software Corporation, www.Alphasoftware.com. Details about the offline features are provided as part of the release notes for the latest release.