Leading SoftwAre MaNiACs®

Cloudbusting

Concept Proposal

EMBEDDED SYSTEMS CAN NOW SYNCHRONIZE DATA WITH THE CLOUD

Table of Contents

Version	Date	Notes
V0.99	2018-05-30	Started
V1.0	2018-06-18	Original draft submitted to XYZ for consideration

Version history

Abstract

Can a reliable file system offer more possibilities for company growth? From an outsider's perspective, growth opportunities can be described as:

Industry-specific redirection and focus: Taking XYZ into more regulatory businesses (automotive, medical device, IIoT, aerospace, …) necessitates more stringent, company-wide processes. The combination of an existing products (like XYZ) with a very lean agile environment has worked well for XYZ. Entering these specific markets requires investing time and energy comprehensive processes focused on safety, risk, and quality management. XYZ has proven the ability to exercise additional controls with RXYZ conforming to MISRA C.

Broadening a device-specific data solution that supports the enterprise: The more file system services that can be encapsulated, the faster it is for an applications developer to deliver a total data solution. This has two immediate benefits. Software maintenance becomes easier to manage and technology improvements can be made "under the covers" with little to no impact to the application.

Add more capabilities to existing toolkits: As flash memory technology becomes less reliable and more intelligence is built into the part itself (ex: managed NAND), there is always an opportunity to add features to XYZ toolkits. But investment in more feature development may not guarantee market growth for XYZ. This is true for established toolkits like XYZ and XYZ.

This proposal applies to only option 2 that explores how a file system is being used through the entire data lifecycle. As data is generated and collected on the device, that data needs to be extracted usually through a separate set of processes and communication interfaces. Data must also be initially loaded in a pre-built image or a real-time download procedure.

That is where Cloud file sharing, already a *technology enabler* for web, mobile and desktop platforms, can become a positive impact to embedded platforms. As data has traditionally resided on a proprietary server has now become displaced with the rapid accent of Cloud file-sharing.

Originally proposed several years ago as *SmartFolder*, a XY Cloud-enabled file system will be referred to as *Cloudbusting* [*].

[*] Tribute to Kate Bush's *Cloudbusting*

Background

The Cloud File Sharing Landscape

File sharing in the Cloud has enjoyed tremendous success over the last few years. Box earned $140M in Q1FY2019 with 85,000 businesses and [1] Dropbox went public in early 2018 with a value of $11 billion with over 500 million users. [2] They aren't the only ones providing cloud-based file services. Vendors include ShareFile (Citrix), Anchor (eFolder), OneDrive (Microsoft), SugarSync, and Google Drive (Google).

Anyone with a browser (or an app) can use these services to upload and download files with others. Some platforms extend their OS file system to automatically by syncing local files with the Cloud. Shared files are then updated on other systems with access privileges to that same data. It is the latter use of a local-OS sync "adapter" plug-in that provides this seamless "magic."

Most Cloud file-sharing services provide automatic sync support with *Cloud file-sharing extension* code on traditional desktop platforms (Windows File Explorer and macOS Finder). For other platforms (like iOS and Android), file-sharing services use specialized apps. Mobile environments are not as automatic since users request access to shared files with these apps. [3]

Currently, no Cloud vendor appears to be focused on these specialized environments. And yet, the need to effortlessly exchange and synchronize data to hundreds, if not millions, of embedded devices is there. And it is the OS file system extension that offers the most promise and the most logical way to extend an embedded file system, like XYZ.

Technology eventually becomes ubiquitous

The fact that embedded file systems don't currently offer Cloud file synchronization may be a concern.

There is a platform hierarchy of capability that moves "downstream" to acceptance. What was common on a server or a desktop computer has become commonplace on mobile devices. Said another way: As mainstream platforms go, target-specific platforms (INTEGRITY, VXWORKS, FreeRTOS, ...) should naturally follow.

It is my opinion that Cloudbusting capability is needed for all embedded platforms.

The Cloud-Connected Embedded Opportunity

The more "mobile" (or more "embedded") the environment, the more complex it is to always keep the cloud and local file system in sync. This is largely due to unstable and unreliable network connections.

Synchronizing even when internet connectivity isn't reliable

One of the benefits of Cloud file-sharing extension add-ons is that, in addition to synchronizing data, it manages establishing and reestablishing connections to the network (the Cloud). The user (and even the local file system) is only aware that file synchronization is either up to date or in progress of being updated.

Embedded systems typically collect information or need to be reconfigured with central systems all based on data transfer. Embedded applications have to either physically connect or use some non-traditional approach to connect and transfer information with host systems.

In my quick review of the market, *de facto* file systems (both commercial and those included in the RTOS) leave data exchange to the customer (OEM) to solve—usually with proprietary technology consisting of proprietary control and non-proprietary protocols (ex: FTP with a custom app or scripts). This archaic approach is cumbersome and possibly difficult to support. To make matters worse, data exchange usually requires user interaction is rarely automatically performed once a network connection can be established.

What might the customer think?

In addition providing to reliable file management services, a customer may not "make the connection" that transfer of file data to "the outside world" is even a possibility on an embedded platform.

I would consider Cloudbusting a technology disruptor by providing an interface into an existing Cloud file-sharing service (like Dropbox or Box). This offering would have a unique marketing spin to it:

"A file system that considers the entire data lifecycle and not just local file system access on the device."

XYZ Already Has a Solid Foundation

XYZ offers the RXYZ reliable file system to solve the inherent risks of data loss associated with modern flash memory. With its established history of reliable flash-aware file management, XYZ provides an outstanding framework to build upon.

To summarize, Cloudbusting is the Cloud file-sharing extension to be offered as an add-on capability to the XYZ file system.

Cloud File-Sharing Interface Toolkits

File sharing providers employ extensive support to encourage third-party development into their extensive facilities:

Company * (Product)	Primary use case	Developer service and use
Box [4]	Business	Annual BoxWorks conference in August
DropBox [5]	Consumer	Extensive developer support
eFolder (Anchor) [6]	Secure business	Currently used by XYZ internally

* Perhaps one or more of these vendors (above) have the ability to help in the investment or marketing of Cloudbusting to further expand their platform reach.

To attract a variety of custom applications, several different ways to connect are usually provided. Vendors provide toolkits that rely on a combination of interfaces of APIs, Internet connectivity interfaces (HTML 5), and scripts (JSON). For most modern-day, robust embedded RTOSes, these standard interfaces should be available.

Why Cloud File Sharing Makes Sense

File sharing over an Internet connection isn't new. However, a File Explorer/Finder Cloud file-sharing extension makes file sharing "transparent" and appear integrated. Because files in the Cloud look like local files on your local system, there is no need for user retraining or app rewriting to access them:

1. Set up a Cloud file-sharing account.
2. Configure what folders that will "live" in the Cloud. Authorize any other user (usually by email address) who can access these folders.
3. Any files created, changed, or removed into that folder hierarchy and files are automatically synced to the cloud. (That goes for subfolders, too.)
4. Consequently, any users (or systems) authorized to share those same Cloud folders are automatically updated with background synchronization.

A Cloud vendor's perspective

"Cloud-based file sharing has transformed how data is distributed and shared among users/devices."

— Box's chief product officer, Jeetu Patel: [7]

The User Benefit

Once a user trials Cloud file sharing, low-cost subscriptions are an easy next step to commit to for users needing an easy way to share large data files, avoiding issues with problematic email attachments. Inherent benefits of security, version recovery, and tracking access make adoption a no-brainer.

The Enterprise Benefit

Further partnerships could be possible with enterprise solutions that manage information between the Cloud and hundreds—if not thousands—of devices:

- Bsquare's DataV Cloud-based data aggregator that is configured to drive business outcomes. [8]
- Siemens MindSphere provides a secure data hosting platform that acquires and transfers industrial data assets. [9]

A Word About Cloudbusting

Cloudbusting assumes that XYZ would need to design and build file sharing access to the XYZ file system. The benefit the user and to apps could set a new standard of data collaboration. This is all due to the capability to synchronize file data with a mirrored file system in the Cloud.

Because XYZ "owns" the file system, the company is uniquely positioned to provide this capability to its customers. This Cloud file-sharing extension bundled with XYZ could distinguish XYZ's toolkit offering from any *de facto* file system available on an embedded platform.

Using a web browser interface to the Cloud won't work either. It must be programmatic since many embedded devices do not support end-user interactions. Already successfully deployed in desktop and server systems (Windows, macOS, ...), embedded systems could benefit with this same capability.

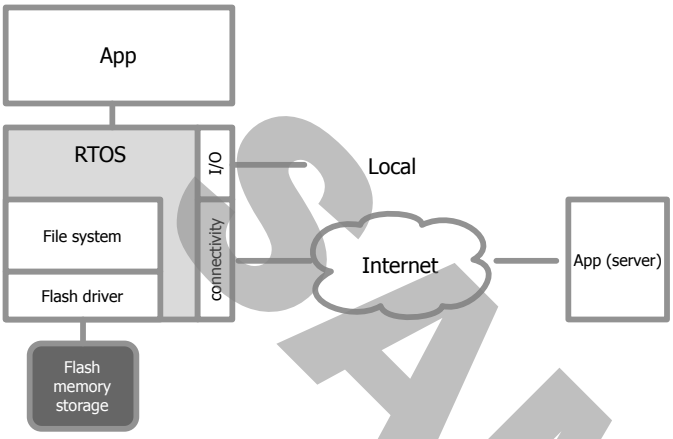
Cloudbusting manages changes to files and subfolders residing within a folder(s) designated to be synchronized with a mirrored folder hierarchy residing in the Cloud. The code would have file synchronization code as well as hooks into the system's connectivity subsystem. Thankfully, Cloud file system vendors provide comprehensive toolkits that can be ported by companies like XYZ. [5]

Let's take a simplified set of steps to show how this all might work. (My apologies to the Datalight developers in advance.)

Design

The Basics

Taking a data view only, key components on an embedded system are shown below (not to scale):



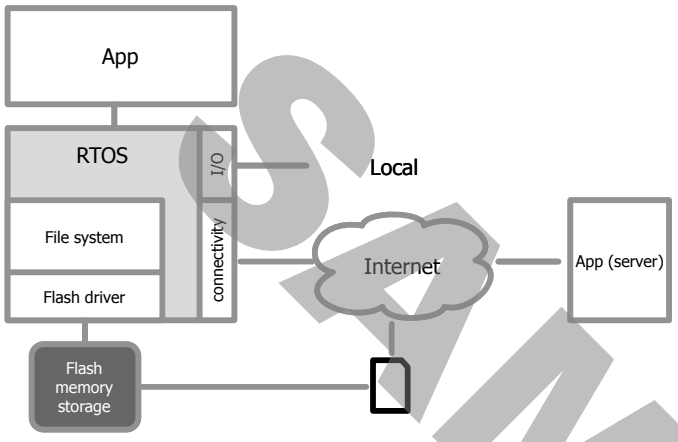
Key components involving data on an embedded system

An app performs a variety of device-specific tasks by utilizing the services of the RTOS. In the case of file data, a file system relies on a flash driver (relying on a simple block I/O interface) that formats, reads, and writes data to permanent, flash memory. As some form of I/O is required to interface to the "outside world" (Bluetooth, WiFi, or direct connected) for transferring valuable data through the Internet. This data on remote systems could, in turn, be managed by apps usually residing on servers.

Here Come the Drones

In preparation of this concept presentation, I met with JD Claridge, CEO of high-rising xCraft, xCraft.io, to discuss how data management is handled on their popular commercial drones. For those Shark Tank fans, you may remember xCraft was one of the few presenters who won investments from all five sharks. [10]

Depending on the industry and the design of the embedded device, data retrieval (and loading) can be cumbersome. In the case of drones, the most typical use case requires a manual effort of copying data to removable flash:



Manual data retrieval for further processing from data collected from drones

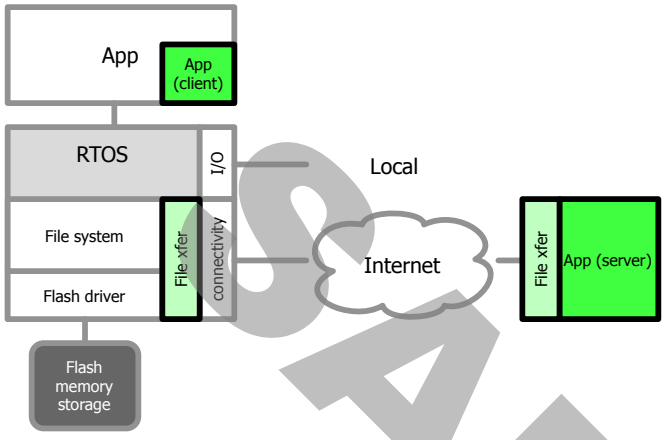
This removable data consisting of pictures, geocoding, fight patterns, temperature sensing data) is then typically copied onto a system for further use. In the case of drones, this information can be used to provide data-centric analysis for farming, first responder, movie making, and other creative uses. The information collected on the device becomes an information collector.

This manual procedure is certainly archaic and commercial applications are requiring more real-time data extraction.

Of course the world isn't run by drones but it is interesting how archaic data collection and data transfer processes are. This is where wireless connectivity and the use of the Cloud becomes a better alternative.

Customizing Solutions to Move Data

As manual techniques are not efficient to loading and unloading data, embedded systems develop custom solutions that require specialized software. This is denoted as App (client) in the figure. File transfer (file xfer) systems code would also be required for the very purpose of moving data using built-in connectivity hardware interfaces to the Internet:



Providing a custom app to move data with file transfer with the Internet

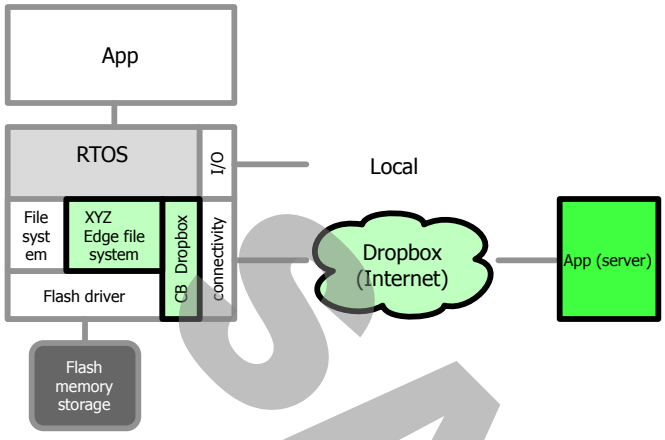
Even if the software isn't proprietary, the complexity associated with establishing reliable connections and ensuring that data is selected, properly secure, and fully transferred can be a challenge, especially for a specialized application. This approach assumes a corresponding file transfer component and associated server app that manages and works with the data transferred from the many devices it serves.

Once this connected file transfer end-to-end methodology works, it probably isn't touched or modified. Fingers are crossed that this file transfer capability works (most of the time).

There must be a better way ...

Providing a Cloud-Based Interface to File Sharing

Alternatively, an “image” can be preconfigured that specific folders are under control of cloud-based interface (Dropbox shown):



Extending the file system with a cloud-based interface

The file system (with Cloudbusting) takes care of the file data synchronization automatically once a connection is established. Even if a connection is dropped (ex: out of WiFi range), the extension includes the logic to reestablish a connection and continue with an interrupted synchronization process.

Comparing with a Traditional Workflow

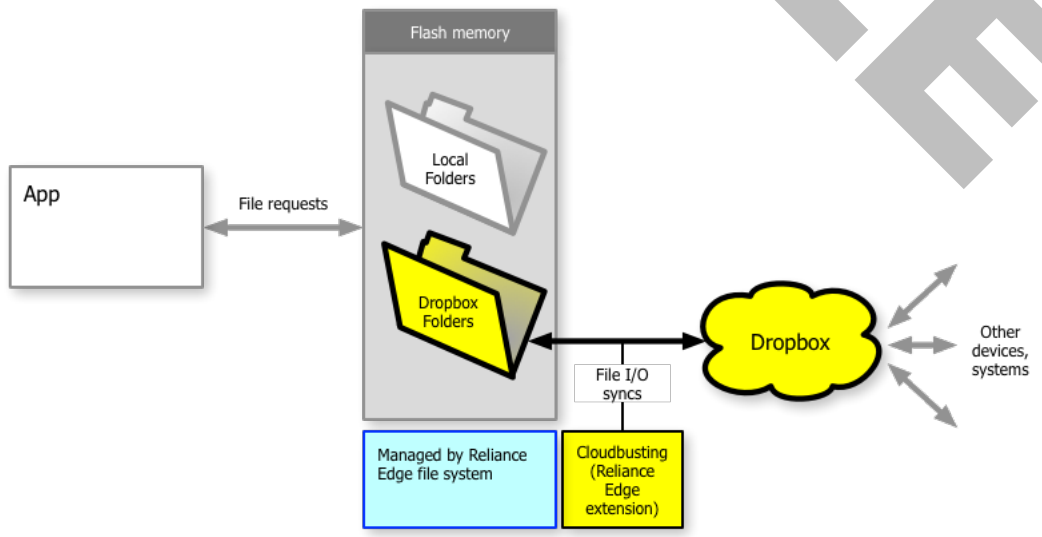
Interpreting the previous diagrams, this table compares approaches to file exchange mechanisms:

Component/Logic	Typical Work	With Cloudbusting
Configure the web interface	Yes	Yes, and set up folders to be shared in the Cloud (Dropbox folders)
App code to manage file transfer	Yes, must establish a connection, move data to be sent/received with file transfer system, manage connection,	Built in. Let Cloudbusting (Dropbox extension module) synchronize with the Cloud in the background
File system	Any, including XYZ	XYZ + Cloudbusting (Dropbox extension module)
File transfer system	Yes (usually something like FTP)	N/A
Overall complexity	✖ Complex and usually custom code (that was possibly developed years ago and nobody wants to touch it)	✓ Couldn't be easier for the customer

Comparing the amount of work and complexity in a typical environment with Cloudbusting

Walk Through a File Request

This diagram should demonstrate the reduction of burden for apps wanting to move data between the embedded device and the Cloud's file-sharing service using Cloudbusting and Dropbox:



Cloudbusting can simplify sharing of data between the embedded device and the Cloud

Here are the steps:

1. Upon system startup, the file system (XYZ) is initialized and Cloudbusting are both initialized.
   Cloudbusting should be already configured to test for new data (subfolders and files) that aren't mirrored between the device's flash memory and the Cloud.
2. The app creates a new file located in the folder designated for mirroring with Dropbox.
   The same applies to multiple files and subfolders and supports deleting, creating, updating, and moving file activities.
3. Cloudbusting, when awoken by the RTOS process scheduler, notices the new files and prepares to synchronize over the Internet to the Cloud.
   Establishment and re-establishment of connectivity is performed in the background.
4. Cloudbusting will use Dropbox programmatic services to verify the state of the Cloud Dropbox file system and transfers the new file to the Cloud.
   In similar fashion, other devices who are authorized to access this same Dropbox shared folder are automatically updated.

Control and Configuration

There are some key programmatic and configuration services that would need to be supported by the Cloudbusting extension:

- Establish and identify folder(s) to synchronize with the Cloud file sharing services
- Start, suspend (pause), and resume background synchronization
- Set how often synchronization should be checked
- Remove folder(s) from being synchronized and add folder(s) to be synchronized
- Manage file/folder renames and moves (optimize removes and adds)
- Track notifications and status changes with synchronization

Thoughts on Building a Cloudbusting Extension to the File System

Tools required to build a Cloudbusting extension aren't solely based on the typical C language designs. To complement C code, there will need to be extensive use of Web languages and scripts. The approaches depend on developer kits supplied by the Cloud file-sharing vendor (Dropbox, Amazon Web Services, Box, and so on).

After a quick review of the Dropbox developer kit, there appears to be ample code examples and a robust developer community to support this effort. (Further information is described in the previous section, "File-Sharing Integration Toolkit Availability.")

All-in-all, a feasibility study would have to ensure that the available tools in a Cloud file-sharing developer kit support services available for the embedded RTOS.

Note: Some embedded platforms may erroneously "appear" to be already supported. The Android Dropbox capability (as with iOS) may be an app requiring user intervention and action. Cloudbusting would instead be a background, "almost silent" programmatic way to synchronize via intercept file requests and availability of Cloud data.

The effort in a development project like Cloudbusting is definitely multi-month and requires specific file system and Web interface skills. Easy to implement? Probably not. Exciting capability worth investing in to extend the usefulness of a proprietary file system (like XYZ)? Probably.

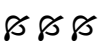
What's Next

Customer need should be evaluated before proceeding on investigating something of the scale to develop Cloudbusting. Obviously, there is no reason to invest in something like this if the return on investment isn't there. There is always the risk that existing device/host file data exchange approaches "do an adequate job." Cloudbusting may be more useful for only new, ground-up customer designs.

It is logical to provide Cloudbusting extension support for XYZ's own XYZ toolkit. However, XYZ could consider writing Cloudbusting to be implemented on other file systems. There just wouldn't be the inherent "reliability" story that XYZ's toolkits have.

Is Cloudbusting a technology solution in search of a problem? It might be worth a phone call to approach Cloud file-sharing companies with robust partner and investor programs (like Box and Dropbox). These vendors may have made the decision to avoid embedded platforms altogether. This could be due to lack of knowledge, resources, or technical experience. They might raise the importance of embedded client interfaces if they knew a company like XYZ could give a new breed of customers access to their Cloud file-sharing services.

Food for thought, I guess.



Reference

1. Condon, Stephanie. “Box Posts Record Revenue in Q1.” ZDNet. May 30, 2018. https://www.zdnet.com/article/box-posts-record-revenue-in-q1.
2. Fiegerman, Seth. “Dropbox Pops in Wall Street Debut.” CNN Money. March 23, 2018. http://money.cnn.com/2018/03/23/news/companies/dropbox-goes-public/index.html.
3. Muchmore, Michael and Jill Duffy. “The Best Cloud Storage and File-Sharing Services of 2018.” PC magazine reviews. January 23, 2018. https://www.pcmag.com/roundup/306323/the-best-cloud-storage-providers-and-file-syncing-services.
4. Box developer. “Build with the Box Platform.” Box. https://developer.box.com.
5. Dropbox Developer. “Build Your App on the DBX Platform.” Dropbox. https://www.dropbox.com/developers.
6. Doring, Anne. “Working with the Anchor API.” eFolder. https://support.efolder.net/hc/en-us/articles/115010471027.
7. Patel, Jeetu. “Are you ready for BoxWorks 2018?.” Boxworks. https://www.box.com/boxworks.
8. Bsquare. “Driving the Next Generation of Industrial Business Outcomes.” Bsquare. https://www.bsquare.com/datav.
9. CXP Group. “MindSphere – Siemens cloud for industry: What is it all about?’ PAC (Blog). May 9, 2016. http://bit.ly/1T6ZfQT.
10. ABC Television Network. “5 Sharks Strike a Deal with xCraft Drones – Shark Tank.” Youtube. October 25, 2015. https://www.youtube.com/watch?v=tmdpF54Hz0Y.