01 / MANAGEMENT BRIEFING

A Complete IoT Solution

Build secure connected products on the hardware your product needs.

Real Time Logic provides a common software foundation from device to service. Choose a suitable embedded platform, including Arm-based designs from vendors such as NXP, and retain your existing firmware investment. Build device applications and remote management on integrated communication, database and security facilities.

ON THE DEVICES
Xedge

Xedge runs device applications above the hardware and native firmware. Xedge32 is its pre-integrated ESP32 option. Other hardware requires target integration and Lua bindings that connect applications to device functions.

ONLINE OR ON PREMISES
Mako Server

Mako is the application server. Run it on an online server or on-site hardware to host the management portal, business logic, SMQ messaging broker and SQLite database.

CONNECTING THE SOLUTION
SMQ

SMQ is the messaging system connecting devices, browsers and Mako. It carries live status, commands and replies, including direct communication with one device.

SQLite is the built-in database that stores customer, device and service records on the Mako server. LSP-Claw helps an approved AI assistant build, run and verify applications on Xedge and Mako.

Put your engineering effort into the product customers buy. Reduce the work of assembling a messaging broker, database server and web framework, then connecting and securing them alongside the device software.

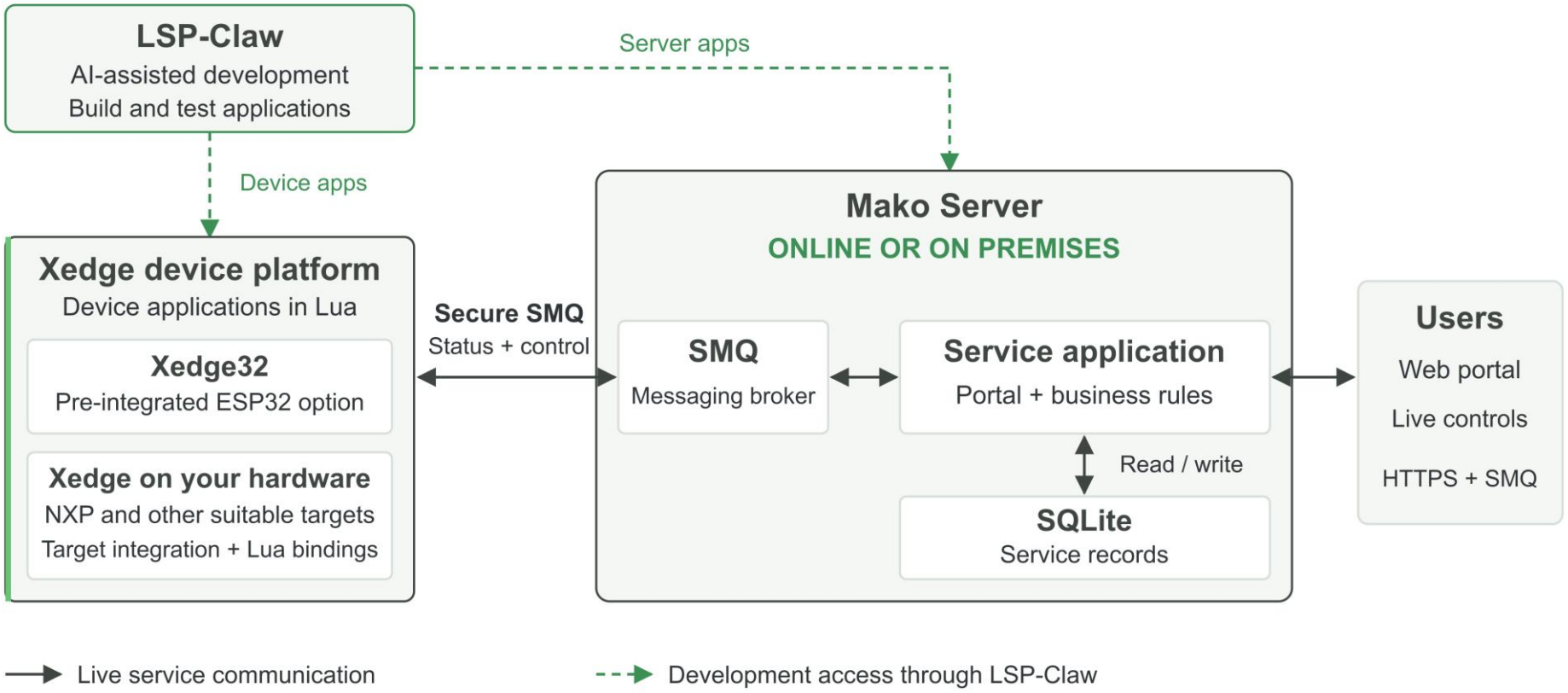
Our recommendation Use Xedge on the devices and Mako for the service. Select Xedge32 for pre-integrated ESP32 support, or integrate Xedge with your chosen hardware. Keep the same service architecture as your product range grows.

Platform basis: [1–4, 7, 11–13, 22–24]. Hardware suitability and integration scope are assessed for the selected target.

02 / MANAGEMENT BRIEFING

How the IoT solution fits together

Xedge powers device applications. Mako runs the service online or on premises, with the SMQ broker and SQLite database in the same server environment.



Both device paths use the same service architecture. LSP-Claw supports application development after the target and required hardware interfaces are integrated; the next page explains that engineering work.

Sources: [1–4, 7, 11–14, 22–24]. SMQ routes messages through Mako. SQLite stores service records on the server.

03 / MANAGEMENT BRIEFING

Choose your hardware and retain your firmware investment

Xedge supports a common application approach across suitable embedded targets. Choose the integration path that fits your product and engineering team.

Product decision	Xedge32 for ESP32	Xedge on your selected hardware
Hardware fit	Products built around supported ESP32 hardware.	Products using suitable NXP, other Arm-based or other embedded platforms.
What is already integrated	The runtime and supported hardware interfaces are exposed to Lua.	The Xedge application platform and its services. Existing ports can provide a starting point.
Engineering needed	Use the supplied bindings. Add native integration for any unsupported peripheral or product function.	Integrate Xedge with the target system and expose drivers and product functions through Lua bindings.
Application development	Many applications can be built in Lua without additional C coding when supplied bindings cover their needs.	After integration, develop higher-level features in Lua while retaining native drivers and timing-critical code.

What integration means

First, make Xedge run with the selected board, operating system, networking and storage. Then add Lua bindings: interfaces that let Lua applications call your existing firmware functions. A working board port does not automatically provide bindings for every sensor, actuator or product feature.

Xedge4Zephyr provides a documented starting point for Zephyr projects [24]. Confirm target resources, required bindings and device security as part of the integration plan.

> **Protect the application investment** Keep the Lua interface consistent as hardware changes, so suitable application logic can be reused. Both paths retain Mako, SMQ and SQLite as the service foundation.

Sources: [11, 12, 22–24]. Target integration is engineering work; reuse depends on compatible application interfaces and hardware capabilities.

04 / MANAGEMENT BRIEFING

Put engineering effort into the product customers buy

Every connection between a separate broker, database server and web framework becomes integration, security and maintenance work your company must own. Mako brings these facilities together.

Component assembled separately	Work the product team still carries	What the RTL approach changes
MQTT broker	Connect messages to customer accounts, device commands, web screens and business rules.	SMQ runs inside the application server and supports direct device communication.
SQL database server	Configure a database service, manage its credentials and connect the web application to it.	Built-in SQLite stores records locally without a separate database service.
Web application framework	Add secure serving, live messaging, database access and device integration.	Mako provides those facilities in the same runtime as the service logic.
The assembled deployment	Coordinate configuration, upgrades, failure diagnosis and recovery across service boundaries.	Operate a compact application deployment with fewer independently configured parts.

One virtual machine does not remove separate services

Putting the broker, database server and web application on the same machine reduces hosting sprawl. It does not remove their interfaces, credentials or upgrade dependencies. Mako removes much of this assembly work at the software level.

> **The cost management should compare** Count development, integration testing, operational support and maintenance over the product lifetime. A component with no purchase price can still create substantial work around it.

Sources: [1–3, 8]. Integration and ownership consequences are management analysis of the two deployment models.

05 / MANAGEMENT BRIEFING

Complete a device management action in one service

Select a device. Send a command. See the result. An operator changes a device's reporting interval from five minutes to one minute, using the company's branded web portal.

Step	What the operator experiences	How the RTL solution supports it
1 Select the device	Sign in and choose equipment assigned to the customer account.	Mako hosts the portal and login. The application uses SQLite records to check ownership and display permitted devices.
2 Authorize the change	Request the new reporting interval.	Mako checks the operator, customer, target device and permitted setting before the command is sent.
3 Reach the device	The request reaches the selected equipment.	SMQ sends directly to the connected device. Its Xedge application validates and applies the setting through the integrated device interfaces, then replies.
4 Confirm the result	See the confirmed setting on the screen.	The application matches the device's reply to the request. SMQ carries the result back and updates authorized browser screens.
5 Keep the history	Review who changed the setting and what happened.	The application records the operator, device, time and outcome in SQLite, making the history available through Mako.

If the device does not reply, the application shows a pending or failed request. A command is marked complete only when the device confirms the result.

The complete solution advantage Mako supplies the portal, messaging broker, application services and database access together. Your team builds the product workflow on this foundation. Run the same service online or on premises.

Sources: [1–8, 11, 12]. This example describes application behavior built with the platform. Product rules, command tracking, timeouts and completion checks are implemented in the service.

06 / MANAGEMENT BRIEFING

SMQ makes a conversation with one device simple

Imagine an operator changing one device's reporting interval. The service must reach that device, receive its answer and show the result to the right operator.

MQTT sends messages to named channels called topics. To make this behave like a direct conversation, the application must arrange a command channel, a return channel and rules that connect the reply to the request. SMQ gives every connected participant an address and carries the sender's address with the message.

Step	MQTT direct messaging workaround	Native SMQ direct messaging
Send the command	Publish to a topic that the application has designated for that device.	Send to that connected device's address through the broker.
Return the answer	Agree where replies go and arrange the necessary subscriptions. MQTT 5 adds response information, but still routes through topics.	Reply to the sender address supplied with the message. No separate reply topic needs to be invented.
Detect a lost connection	Arrange status messages that tell the application when a device connects or drops out.	Ask the broker to watch that device's connection and notify the application when it disconnects.

Why this matters Direct interaction is routine in device management. SMQ makes it part of the communication model, removing a layer of routing conventions and supporting code that MQTT leaves to the application.

The same model works between a device, a browser and server logic. Broadcast updates remain available when several users need the same information.

Sources: [3–6, 10]. SMQ messages still pass through the broker. A connection address changes on reconnect; product identity, command tracking and completion checks remain application responsibilities.

07 / MANAGEMENT BRIEFING

SMQ reduces the work of managing a growing fleet

SMQ combines direct addressing, live browser communication and connection supervision with a broker that is part of the application server.

Management need	What SMQ provides	Business benefit
Add devices and management features	Reuse direct addresses for commands and replies instead of creating separate command and reply channels for each device.	Less custom message-routing code as the product grows.
Keep operators informed	Send live changes to connected screens and supervise device connections.	Operators see current activity and can react to lost connections.
Serve separate customers	Keep each customer's messaging in its own broker within Mako, with a separate web address.	Organize customer-specific messaging without a separate broker service for every customer.
Control who can do what	Reuse a user's web login and check which devices and actions they may access.	Keep device permissions close to the application's customer rules.
Expand server capacity	Use SMQ clustering to distribute messaging across servers.	A supported route to grow messaging capacity and remove a single broker failure point.

The stronger scaling model for device management Add devices without inventing more command and reply channels to simulate direct conversations. SMQ supplies direct messaging, while customer separation and clustering provide additional ways to organize and expand the service.

SMQ also narrows common access mistakes: it has no wildcard subscriptions that can accidentally request an entire tree of topics. Apply explicit permission checks and encrypted connections to control every customer's access.

Sources: [3–6, 15]. The comparison concerns management architecture, not an unmeasured messages-per-second ranking. Broker clustering does not automatically replicate application records or SQLite databases.

08 / MANAGEMENT BRIEFING

SQLite can be the better production choice

For an IoT service whose application server manages access to local records, SQLite can deliver the required database capability with less administration and less communication overhead.

SQLite is a SQL database engine built into the application process. It supports transactions: a set of related record changes can succeed together or be rolled back together. You keep structured queries and reliable records while removing a separately operated database server. [7–9]

Benefit	Why it matters in an IoT service
No separate database service	No additional database server installation, network listener, service account or client-to-server database connection to manage.
Local database calls	The application reads and writes records directly. It avoids sending each query to another process or over a network.
A practical fit for service records	Device inventory, customer settings, permissions and event history can live alongside the service that owns them.
Straightforward data management	Use standard SQL, indexes and transactions, with a compact database file and supported backup tools.

The performance case goes beyond simplicity

SQLite's own guidance reports that it is often faster than a client/server SQL engine when used behind an application-specific server. This supports choosing SQLite deliberately for this architecture, rather than treating it as a temporary development database. Actual throughput depends on the workload. [7]

Many devices can share the service Devices talk to Mako, not directly to the database file. Mako manages their requests and keeps database updates short. SQLite permits one writer at a time per file; this does not mean only one user or device can use the service.

Sources: [7–9, 16–18]. Keep database files on local storage, validate peak write demand and use a consistent backup procedure. The benefit is workload-specific, not a universal database speed ranking.

09 / MANAGEMENT BRIEFING

Deploy the whole service on a compact server

Deploy Mako with SMQ and SQLite on a low-cost online virtual private server, or on your own on-premises hardware. Xedge-powered devices connect to that service.

ONE SERVER ENVIRONMENT

Run Mako as a managed operating-system service. Deploy the application with its web pages, service logic and SMQ broker configuration.

LOCAL BUSINESS RECORDS

Keep SQLite on local server storage. Protect the data with scheduled consistent backups and a tested restoration procedure.

ONE SECURE WEB ENTRY

Use HTTPS for the portal and SMQ connections. The messaging broker shares the web service rather than requiring a separate public broker port.

Fewer services make operation easier to understand

The team has fewer processes to provision, fewer connections between services to troubleshoot and fewer configurations to coordinate. A release can package the service behavior coherently instead of relying on a sequence of independently configured infrastructure changes.

Keep control of deployment and customer data

The same approach can run on a hosted virtual server, an on-premises computer or an embedded Linux gateway. The company chooses the hosting provider, data location, release schedule and customer-service model. A managed local deployment can also keep site services close to the equipment.

Budget for the service you operate Size the server against active connections, message traffic, database writes and retained history. Include hosting, licensing, support and recovery in the operating budget. The structural saving is fewer separate services to assemble and maintain.

Sources: [1–3, 8, 14, 18]. A single server is a compact deployment, not automatic high availability. Plan recovery and expansion to match the service commitment.

Protect device identity and contain a breach

A stolen device credential should never become a pass to the whole fleet. Xedge, SMQ and Mako provide complementary protections on the device and server.

RTL's MQTT security article shows the danger of combining copied device credentials with overly broad messaging permissions. An attacker may appear to be an approved device and then read other traffic or send unauthorized commands. Encryption protects the connection; the service must also control what each connected device can do. [20]

Risk to the product	Protection available in the RTL solution
Copying device credentials	Xedge provides softTPM facilities that keep private keys out of ordinary application code. Device-bound identity requires provisioning and hardening for the selected hardware, including on Xedge32.
Reading messages across the fleet	SMQ has no wildcard subscriptions. A client cannot use a wildcard to request all topic channels. Explicit permission checks control which messages it can receive.
Misusing a valid device identity	Mako and SMQ let the application check the customer, destination device and permitted action before forwarding a message. Separate customer brokers help contain access within each customer's service.
Exposing credentials in a web page	SMQ can reuse the browser's authenticated web session. The application avoids placing a long-lived device password in downloadable browser code.

The integrated security advantage Protect credentials on the device and enforce permissions in the server that understands the customer. RTL supplies both foundations together, reducing the security integration work left to the product team.

AI can build higher-level product behavior using existing encrypted communication, protected-key services and signed application capabilities. For each hardware target, engineering must configure and validate key protection, boot and update security.

Sources: [5, 6, 11, 12, 20–22]. Production protection requires device-specific provisioning, hardening, enforced permissions and credential revocation. A valid identity does not by itself authorize an action.

Control which devices and users can act

A valid identity should grant only the access needed for its assigned job. Enforce permissions before a command reaches a device.

Consider a service used by Customer A and Customer B. Each has its own devices and operators. In this example application, Mako checks authenticated identities against server-held ownership and permission records; a customer or device name supplied in a message does not establish permission.

Request in the example service	Decision enforced by the application	What the customer gains
An authorized operator changes a setting on Customer A's device.	Allow the permitted setting change, send it through SMQ and record the device's confirmed response in SQLite.	Useful remote control with a clear record of the outcome.
Someone with Customer A's valid credential targets Customer B's device.	Reject the request before forwarding it. The authenticated account does not own or have permission to control that device.	Another customer's equipment remains outside the account's reach.
A device allowed to report readings tries to command other equipment.	Reject the action. Permission to send readings does not include permission to control another device.	A compromised reporting identity has restricted authority.
A valid operator requests a setting outside the product's permitted range.	Reject the value before sending the command; the device application also checks its supported settings.	Product limits remain in force during remote operation.

Why integration matters The SMQ broker runs beside the application rules and customer records in Mako. The service can check who is acting, which device is targeted and what action is allowed in one environment.

Demonstrate both the allowed change and a rejected request for another customer's device. Show the rejection in the service history and confirm that the other device's setting remains unchanged.

Sources: [1, 3, 5, 6, 8]. These are example policies implemented with Mako and SMQ authorization controls, not automatic permissions. Apply checks to every relevant message path and use server-controlled identity and ownership records.

Use less AI effort to deliver working features

LSP-Claw and Xedge reduce the infrastructure AI must generate and give it feedback from the running application.

Tokens are the units an AI service processes and often charges for. With Xedge, compact Lua applications call existing web, messaging and security services. Compared with generating equivalent infrastructure in C, there is less supporting code to describe, generate and review.

REUSE THE PLATFORM

Build features above existing services and integrated hardware interfaces. Xedge32 supplies bindings for supported ESP32 hardware; other targets need their own integration.

TEST WHILE BUILDING

LSP-Claw lets an approved AI assistant run the application, test its web interfaces and inspect diagnostic output in the development environment.

CARRY SKILLS ACROSS TARGETS

Xedge, including Xedge32, and Mako share the Barracuda App Server foundation. Reuse Lua skills and suitable modules across device and server work.

Describe It. Build It. Run It. Verify It. The deliverable is a feature that runs and can be checked. Runtime feedback helps the assistant correct mistakes without relying only on guesses about generated code.

Make the AI contribution visible

Demonstrate the same device-management interface: describe the required screen and setting change, let the assistant build it through LSP-Claw, run it, inspect the response and correct an issue. Show the application working and retain the test evidence.

A direct route to lower development cost

Less generated infrastructure and shorter debugging loops create a strong basis for lower token use. For other hardware, budget the initial port and Lua bindings separately, then measure application-development savings. Track tokens, engineer time and rework per accepted feature; engineers retain validation and release responsibility.

Sources: [2, 11–13, 19, 22–24]. Lower token use is an architectural expectation, not a published percentage benchmark. Keep AI development access and credentials separate from production.

Build the service your customers will pay for

Choose Xedge for device applications and Mako, SMQ and SQLite for the service, with LSP-Claw supporting development. Keep hardware choice within your product strategy.

Product investment	What the customer or company gains
A consistent branded portal	Customers see their equipment, status and service records in one place.
Remote configuration and diagnostics	Support staff can investigate and assist through controlled device communication, reducing dependence on site visits where remote resolution is possible.
Reusable customer and device rules	New product models and customer groups can build on the same service foundation. Consistent device interfaces support reuse across hardware generations.
Company-controlled deployment	Choose hosted or local services and keep control of customer data, releases and the commercial offer.

Customer example Graviti Energy

Real Time Logic designed a system for remotely restarting Graviti Energy's EV charging stations, using custom hardware, Xedge32 (the pre-integrated ESP32 edition of Xedge), SMQ and a Mako cloud portal. Device and server applications used Lua, with no custom C code needed for the device application. The solution included remote firmware and application updates. Read the customer case study [25].

Measure the advantage per delivered feature

Record engineer hours, AI tokens, deployment steps and independently configured services. Compare equivalent workflows using the same acceptance tests. Use measured results to substantiate the development and operating benefits.

Recommended next step Select the device hardware and agree its integration scope, licensing, support and acceptance targets. Demonstrate the complete workflow, then add enrollment, updates, diagnostics and recovery on the same foundation.

Platform basis: [1–6, 11–15, 22–24]. Product and commercial outcomes are recommended uses of the platform; customer demand and quantified savings are established by the product program.

Server and messaging references

Primary documentation supporting the platform comparison. Public sources reviewed 17 October 2026.

1 **Mako Server overview**

Integrated Lua application server, web interfaces, SQLite, security and IoT messaging.

2 **Barracuda App Server**

Common application foundation for server and embedded products.

3 **SMQ product overview**

Interactive device management, browser integration and customer-specific broker instances.

4 **SMQ protocol documentation**

Direct addressing, sender return addresses, publish/subscribe and connection supervision.

5 **SMQ authentication and authorization**

Web-session identity, device authentication and explicit message-permission controls.

6 **SMQ broker integration**

Application hooks for connection management, authorization and undeliverable messages.

10 **MQTT version 5 standard**

Request and response features remain based on topics and subscriptions.

15 **SMQ cluster management**

Messaging distribution across servers and the engineering considerations for clustering.

20 **How hackers can penetrate an MQTT solution**

Credential theft and excessive messaging permissions explain why authentication alone is insufficient.

Database and deployment references

7 **Appropriate uses for SQLite**

Application-server use, reported performance advantages and the single-writer model.

8 **SQLite without a separate database server**

Database operations execute within the application process.

9 **SQLite transactions**

Related database changes commit together or are rolled back.

14 **Low-cost Mako and SMQ deployment**

Deploying the broker and application server as a Linux service on a VPS.

16 **SQLite query efficiency**

Local database calls avoid client/server communication overhead.

17 **SQLite write-ahead logging**

Concurrent reading and writing, with one writer at a time per database file.

18 **SQLite backup API**

Supported procedures for consistent backups of an active database.

Xedge hardware and development references

22 **Xedge platform overview**

The embedded application platform, integration with native firmware and the Xedge32 edition.

11 **Xedge32 pre-integrated ESP32 option**

Ready-made hardware bindings and application development for supported ESP32 hardware.

24 **Xedge4Zephyr**

Xedge integration for Zephyr, with build guidance as a starting point for target projects.

23 **Connecting native firmware through Lua bindings**

Interfaces that expose C functions and product hardware to Lua applications.

12 **Using Xedge**

Application lifecycle, configuration protection and development integration.

21 **How the softTPM works**

Protected key operations and device-bound identity, with provisioning and hardening requirements.

13 **LSP-Claw instructions**

Development tools, runtime feedback and controlled access to the running lab.

19 **Lua manual**

The application language and its integration with native C code.

25 **Graviti Energy customer project**

Custom hardware, Xedge32, SMQ and Mako for remote EV charger control and fleet updates.