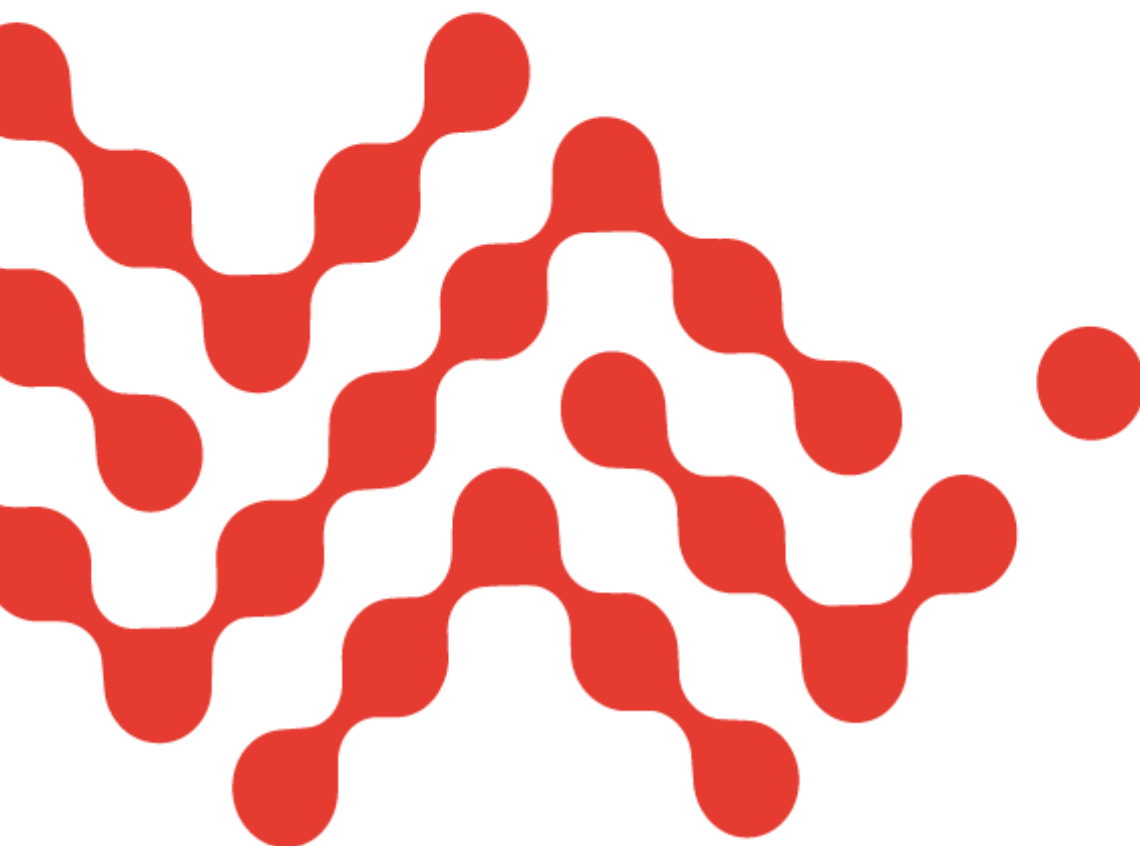


ReadyAgent

Features





Copyright

© 2010 Sierra Wireless. All rights reserved.

Trademarks

AirCard® and Watcher® are registered trademarks of Sierra Wireless. Sierra Wireless™, AirPrime™, AirLink™, AirVantage™ and the Sierra Wireless logo are trademarks of Sierra Wireless.

wavecom®, , , inSIM®, WAVECOM®, WISMO®, Wireless Microprocessor®, Wireless CPU®, Open AT® are filed or registered trademarks of Sierra Wireless S.A. in France and/or in other countries.

Other trademarks are the property of the respective owners.

Document history

Date	Version	Auteur	Description
27/11/08	01/A	Cuero Bugot	First Version
02/04/10	02/A	Cuero Bugot	Added new features of the ReadyAgent
30/06/10	02/B	Cuero Bugot	Minor edits for ReadyAgent R2.0 release
04/10/10	02/C	Cuero Bugot	Add features from ReadyAgent R3.0 release

Content table

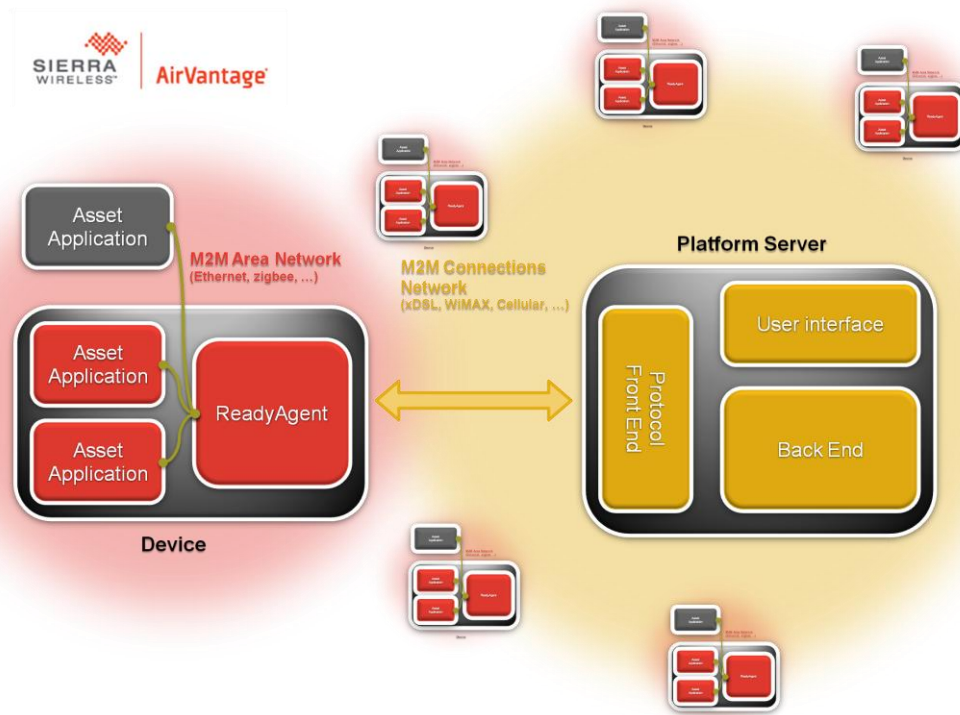
ReadyAgent	1
Features	1
Document history	2
1. Reference documents.....	5
2. Introduction	5
3. Asset management.....	6
4. Device management.....	7
4.1. Monitoring	7
4.2. Software update	7
4.3. Application container.....	8
4.4. Remote script	8
5. Network and connectivity management	8
5.1. Server connection proxy.....	8
5.2. Network manager	9
5.3. Connections policy.....	9
5.4. Wakeup mechanisms.....	9
5.4.1. SMS wakeup	10
5.4.2. Mediation protocol.....	10
6. Maintenance debug and diagnose.....	10
6.1. ReadyAgent configuration	10
6.2. Configuration store	11
6.3. Shells.....	11
6.4. Web Server	11
6.5. Application logs upload	12
7. Security.....	12
8. Other services	13
8.1. SMS API	13
8.2. Data storage API	13
8.3. Time synchronization	13

1. Reference documents

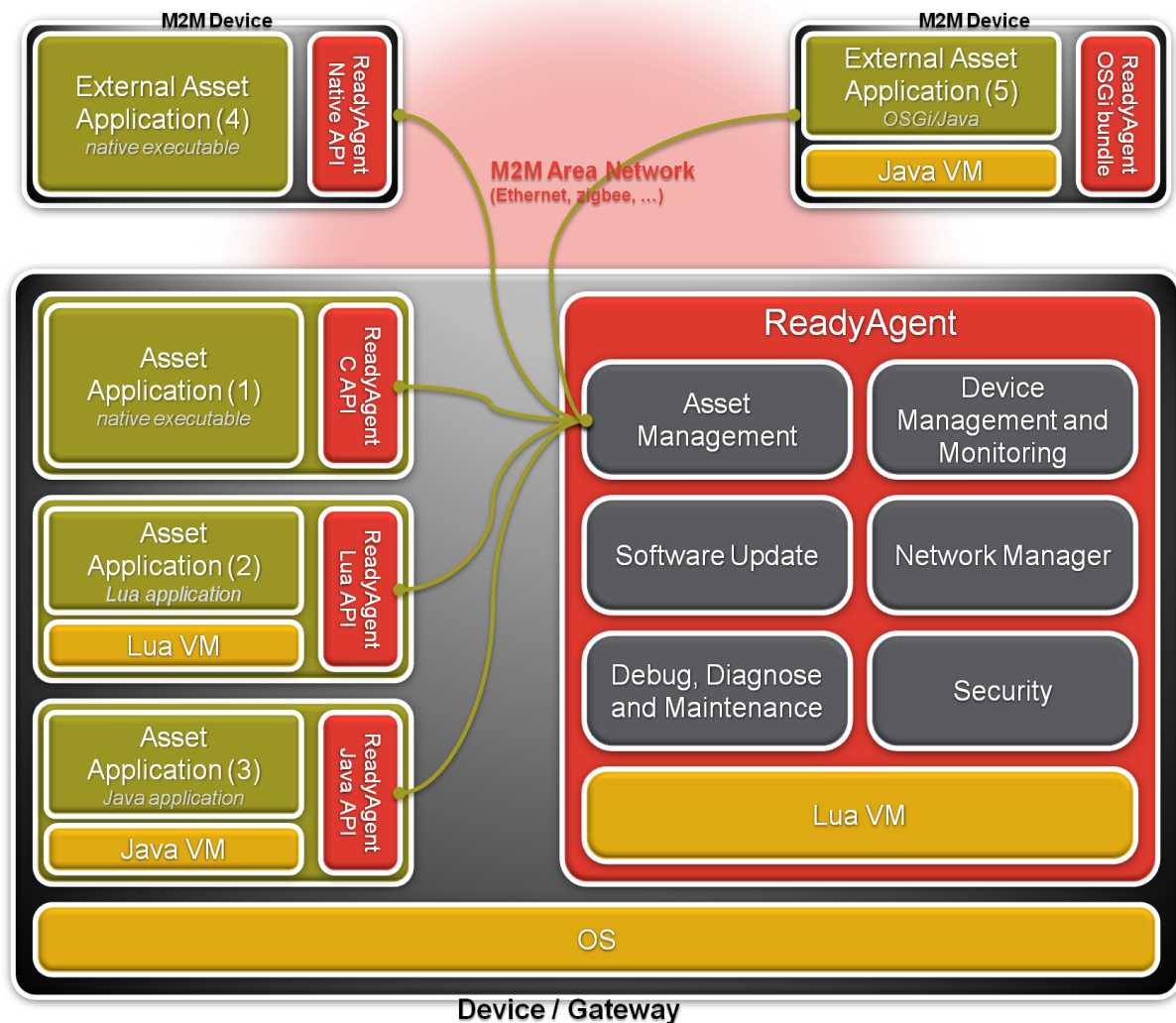
- [REF-1] ReadyAgent User Guide API
- [REF-2] ReadyAgent Configuration
- [REF-3] PLT08 SP01 AWT-DA-Protocol
- [REF-4] Monitoring Engine
- [REF-5] AWT-DA Security extension

2. Introduction

The ReadyAgent is a part of the global Services Platform. The ReadyAgent is a piece of software that runs on the embedded device, and that is in charge of the communication with the platform server.



The ReadyAgent is used as a proxy for the user embedded applications, and it also provides a bunch of services that increase the usability, efficiency, debug and diagnose of the embedded device software.



The same agent is used for all embedded applications. It runs into its own process (for Operating Systems that support processes at least) to ensure a good separation with the applicative code.

This document describes all the high level features of the ReadyAgent.

3. Asset management

Each embedded application links with the ReadyAgent libraries which are actually connectors to the agent process.

The applications can send data, events or register to listeners in order to receive commands or acknowledgements. In terms of Asset Management, the ReadyAgent really act as a proxy or gateway between the assets (business applications) and the Platform Server. This greatly simplifies the business application development by removing all the complexity of the network connection, retries, connection errors, server availability workflows.

The API used to communicate with the agent is detailed in the document [REF 1]. This document also gives a couple of uses cases and code samples.

4. Device management

4.1. Monitoring

The ReadyAgent implements a Device Monitoring feature. The Monitoring Engine is a system that allows connecting a set of predefined *triggers* to customizable *actions*, enabling optional *filters*.

- Triggers can be on variables changes (threshold, deadband, etc.) and system events (on boot, server connection, etc.).
- Actions provided can send data and events, log variables, etc.
- Filters are simple Boolean logic functions.

As a result, the agent can monitor some device variables (battery level, power supply mode, GSM/GPRS signal strength, system UTC data, etc.) and user defined variables and then send events or data when predefined conditions are verified.

A typical use case could be to send a “Boot Event” on device start up; send an additional “Battery Low Event” when the level of the battery goes below some level; log the signal level variable and flush the acquisition on each server connection.

For a complete Monitoring Engine documentation, have a look to [REF-4].

4.2. Software update

The ReadyAgent embed a set of safe and powerful software update functionalities.

On devices that come with IDS Agent (OASIS based firmware), the Software Update feature is managed by the IDS framework and the ReadyAgent can be updated like any other application.

On others devices, the ReadyAgent provides a Software Update feature. The main feature is the ability to update the whole software or each component separately (the upgrade of the device firmware depends on the Operating Systems though).

The update package will be transported either by OMADM client or AWTDA command. Each component can deal with its own update or use the ReadyAgent to process it.

In the first case, the upgrade process is handled directly by the asset application. The application will receive a notification with an upgrade file when it is requested to upgrade. Furthermore, by the mean of this notification, the application can update third party software that is not directly connected to the ReadyAgent.

Applications using OSGi bundles are a special case where the upgrade process is completely transparent for the application: OSGi framework will deal with it.

In the second case, the ReadyAgent will execute a Lua script that contains the process for that update.

This can be useful to update applications that don't come with self-update feature. In that case the user has only to write a simple script that will execute the update of the component.

4.3. Application container

The ReadyAgent can manage others applications thanks to the Application Container module. The application can be a Lua script or whatever application type (on OS that support others type of applications).

Once an application is installed in the Application Container, it can be automatically started on ReadyAgent start up. When an application is started, the ReadyAgent will monitor it and eventually restart it if an unexpected error occurs during the lifetime of the application. In order to use this feature, the user needs to conform to a basic API in the application (start, stop functions mainly).

4.4. Remote script

The Remote Script feature enables to send a Lua script that will be executed remotely within the ReadyAgent.

Thanks to the Services Platform, this script can be executed remotely on several devices simultaneously with a single job.

This feature can be very useful to execute some kind of maintenance or debug actions remotely on several devices.

5. Network and connectivity management

5.1. Server connection proxy

The ReadyAgent acts as a proxy between the asset applications and the Platform Server. As written above, it handles the complexity of the network connection, retries, connection errors, server availability workflows.

As a matter of fact, the agent takes care of the transport of the data from the device to the server. Thus it can also add some value to the transport layer such as binary compression, authentication and encryption.

Several network transport layers are supported: HTTP, TCP over IP, and SMS. Note that some restrictions on the transmission capabilities may apply on some specific transport layers (e.g. cannot send huge amount of data over SMS).

5.2. Network manager

The ReadyAgent provides a module called Network Manager. This entity is in charge of actually mounting the bearer in order to enable the connectivity with the server.

Supported bearers really depend on what the device hardware provides. In the usual case you can use GPRS, 3G, Ethernet, Wi-Fi, RTC, SMS.

In order to use the Network Manager, an ordered list of usable bearers must be established into the ReadyAgent configuration ([REF-2]). When a connection attempt is done then each bearer is tried in order. Additional retry times and retry delay can be added in order to retry several times the same bearer before trying the next one.

The SMS bearer is handled a little differently since it does not provide IP connectivity. Rather the SMS bearer is used as a fallback when all IP bearers have failed, then the ReadyAgent can optionally send the data through concatenated SMS.

Depending on the bearer, additional parameters need to be specified (e.g. for GPRS: APN, user and password; for Ethernet: DHCP, static, etc.)

5.3. Connections policy

The general paradigm of the Platform Server and ReadyAgent solution is that the (HTTP/TCP) connection is always initiated by the device. The reason is that the device is usually located behind one or many NAT routers and firewall allowing only outgoing connections. There are several solutions to this problem, one is the connections policy, and another one is detailed in the next chapter.

The connection policy is a description on how and when the ReadyAgent should initiate a connection to the server. There are four policies that are not exclusive:

- *onboot*: connect when the device reboot.
- *period*: connect every period of time
- *cron*: connect when the cron entry is verified
- *ondemand*: connect when some asset application pushes data to the agent

In addition, any asset application can force at anytime a connection with the server by calling a dedicated API (see [REF-1])

5.4. Wakeup mechanisms

Some times, a server action may require that the device connects to it at an unpredictable time. The wakeup mechanisms allow the server to contact a device at anytime. Several wakeup mechanisms are provided in order to adapt to the different network configurations in the field.

5.4.1. SMS wakeup

This is the most common use case: the Platform server can send SMS in order to tell the device to connect by IP. As a side note the Server can also send commands and configuration data through SMS. This wakeup mechanism requires a GSM modem, and may cost a lot depending on charging of the carrier.

5.4.2. Mediation protocol

The mediation protocol is using UDP and thus is a very lightweight protocol. It enables an efficient wakeup mechanism. The mediation protocol never actually transports data, but can only ask a device to connect to the server.

There are two different use cases for using the mediation protocol:

- It can be used when the device is behind NAT/Firewall routers in device polling mode when the server cannot reach the device with an IP/port. The server can still reach the device in between polling since the reverse route stays open a couple of minutes in the traversed NAT/Firewall routers. This obviously can only be used when bandwidth does not really matters since at least one UDP packet is sent every couple of minutes in order to keep the reverse route open.
- It can be used as a server initiated command when the server can actually reach the device through an IP address/port (UDP), i.e. the device has a public IP (either its own or a NAT routed IP). In this case minimal bandwidth is used since only one packet is sent, and only when the server needs to wake the device up. This enables the mediation protocol even where bandwidth has a cost.

6. Maintenance debug and diagnose

6.1. ReadyAgent configuration

The document [REF 2] offers a detailed view of the configuration capabilities of the agent. The ReadyAgent is shipped with a default configuration, but it can be easily modified locally or remotely.

- Remotely with AWTDA protocol data writing from the server. Data writing can be done over a standard HTTP/TCP connection or in the lightweight SMS extension.
- Locally, by embedded business application (asset application), with the API detailed in [REF-1].
- Locally and manually by AT commands (on modem targets).
- Locally and manually by the Lua shell access.
- Locally and manually by the HTTP Web server embedded into the agent (not available yet).

The configuration can be committed to a persistent storage area in order to keep the same configuration after a reboot. Similarly the default configuration that was shipped with the device can be restored at any time, overwriting the current configuration.

6.2. Configuration store

The ReadyAgent also provides a Configuration Store to local application. The same API as for the agent configuration is provided to the local application in order to store its own configuration. This feature provides local applications with persistent storage and remote access to that configuration from the server side.

In order to use the configuration store, just create a sub branch in the `config.user` tree.

6.3. Shells

The shell service enables the Operating System Shells to be exported. So the type of shell supported (in addition of the Lua shell) actually depends on the Operating System. For instance on a standard Linux box, a UNIX shell can be made available.

Moreover the ReadyAgent provides a Lua shell that enables a powerful introspection of the agent machinery. With the Lua shell you can interact directly with the agent so that you can enable precise debug and diagnose capabilities.

Understandably using the Lua shell to interact with the agent requires the operator to have some knowledge on how the agent runs.

You can access the shells locally by directly connecting to the IP/Port with a telnet or similar program as you would do for a regular shell (telnet or ssh) connection.

If you do not have an IP/Port access (your device is either behind a NAT/Firewall router or not even connected), then you can access the device remotely through the Platform Server.

Note: the *remote* access is not fully available yet.

6.4. Web Server

Note: this feature is not available yet.

The ReadyAgent will provide an embedded Web Server in order to facilitate local accessibility. The Web Server will allow configuring the ReadyAgent, Monitoring Engine, etc., and interacting with the agent itself in order to perform maintenance or debug tasks.

6.5. Logs store and upload

The ReadyAgent provides a mechanism in order to easily store log locally. Moreover the log framework will be able upload, on demand, those logs on the Server, in order to ease diagnose and maintenance tasks.

Each log entry is identified by a module name, a severity, and the log text itself. The module and severity parameters allows to filter the log before they are emitted, allowing precise logging configurability.

Level is one of the following (ordered list): *ERROR*, *WARNING*, *INFO*, *DETAIL*, and *DEBUG*.

The log store is composed of two areas: one is a buffer in RAM to store current logs, and another one is located into a persisted storage in order to survive reboot. The logging framework offers different readily usable policies.

- **context**: store in flash logs that reach the configured level plus the preceding cached log entries coming from ram buffer.
- **sole**: store in flash all logs entries with higher severity than the configured level . No context is stored, only the sole log entry.
- **buffered_all**: each log entry is stored in ram, and then when the ram buffer is full it is stored in flash.

7. Security

Note: Security layer is not fully functional yet.

The ReadyAgent provides a full featured security layer.

The security layer provides three levels of features:

- data integrity
- peer authentication
- data encryption

The two first level of security are actually usually done together since it does not add any overhead. The full security specification is available here: [REF-5].

The security layer is provided as an extension of AWTDA and is fully optional. As an extension it does not require an AWTDA protocol modification but rather adds fields in order to achieve its goal. All the security-related add-ons are located into the transport layer i.e. the Envelope. Additionally the AWTDA security extension defines some specific workflows for setting up secured connections.

The AWTDA authentication scheme was directly inspired from the OMADM authentication scheme. It uses HMAC with MD5 or SHA1 hashing functions, and also uses a nonce in order to prevent from replay attacks.

The AWTDA encryption scheme uses AES ciphering algorithm. Supported profiles are aes-cbc-128 and aes-cbc-256. The cipher key is generated from the stored credentials and a nonce value, making it changing for each session. This also makes the encrypted data

unique for a pair ReadyAgent-Server; meaning no other instance of ReadyAgent will be able to decipher the data, and similarly no other server will be able to decipher the data (unless they know the credentials and the nonce, of course).

The authentication and encryption is reciprocal, meaning that each peer (the ReadyAgent or the Server) authenticate the other one. Thus the ReadyAgent holds two credentials pairs: one to authenticate to the server, and one for authenticating the server.

8. Other services

The ReadyAgent offers some other services that do not fit in the above categories. They are listed here.

8.1. SMS API

The AwtCom library provides a simple to use SMS send and receive API. This allows developing business application that will not depend on what device is used since the API is the same for all hardware targets.

The SMS API handles *Long SMS* splitting and concatenation in order to send or receive more than 140 bytes messages. The SMS API uses 8bits encoded SMS.

The receive API provides a pattern matching entry in order to only receive SMS from specified senders.

8.2. Data storage API

Note: this feature is not available yet.

The ReadyAgent provides a simple API in order to store locally time stamped data so that the asset (business application) does not have to take care of it.

The data can be filtered/decimated locally (as in Round Robin Database) in order to reduce the amount of stored data.

The data can then be sent to the server using the bandwidth efficient AWT-DA protocol.

8.3. Time synchronization

The ReadyAgent can take care of the time synchronization, using the NTP protocol. The agent will synchronize at startup and periodically (configurable) with an NTP server in order to keep the local time as accurate as necessary.

Moreover, the time settings (as daylight saving, time zone) can be provisioned by the Server, so that the ReadyAgent can also provide a localized time.