Envelope Service

User Guide

A111 – Envelope Service

User Guide

Author: Acconeer

Version 1.0: 2018-07-04

Acconeer AB

Table of Contents

1 Introduction

The Envelope Service is one of three services that provides APIs for reading out the radar signal from the Acconeer A111 sensor. The data returned from the envelope service is typically further processed and can be used in different types of algorithms such as distance measurement algorithms, motion detection algorithms, and object positioning algorithms. In use cases where low computation complexity is important, and the exact location of objects is less important you may consider using the power bins service instead of the envelope service. Advanced users that needs phase information for detecting very small variations in distance will probably prefer the IQ data service instead of the envelop service.

Acconeer also intends to provide several easy to use detectors that are implemented on top of the basic data services. The detectors provide APIs for higher level tasks like distance measurements, motion detection etc.

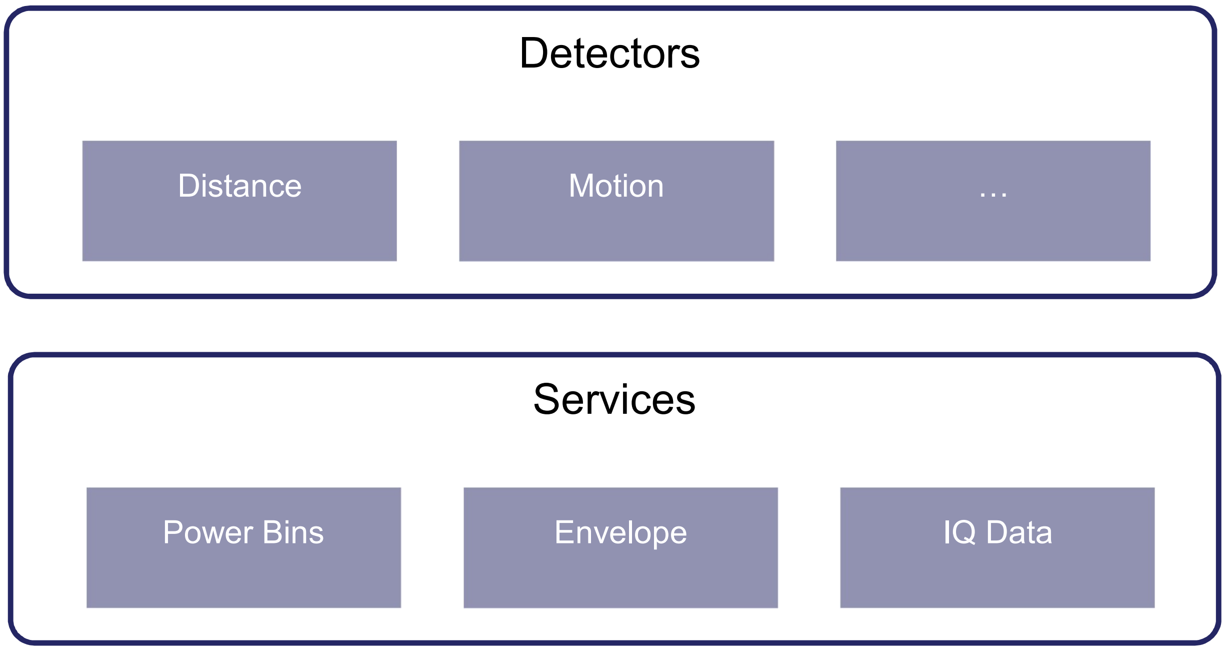


Figure 1- Acconeer SW Interfaces

2 Setting Up the Envelope Service

2.1 Initializing the System

The Radar System Services (RSS) must be activated before any other calls are done to the radar sensor service API.

```
if (!acc_rss_activate()) {
    /* Handle error */
}
```

All services in the Acconeer API are created and activated in two distinct steps. In the first creation step the configuration settings are evaluated and all necessary resources are allocated. If there is some error in the configuration or if there are not enough resources in the system to run the service, the creation step will fail. However, when the creation is successful you can be sure that the second activation step will not fail due to any configuration or resource issues. When the service is activated the radar is activated and the radar data starts to flow from the sensor to the application.

2.2 Envelope Service Configuration

Before the envelope service can be created and activated we must prepare a service configuration. First a configuration is created.

```
acc_service_configuration_t envelope_configuration;

envelope_configuration = acc_service_envelope_configuration_create();

if (envelope_configuration == NULL) {
    /* Handle error */
}
```

The newly created service configuration contains default settings for all configuration parameters and can be passed directly to the `acc_service_create` function. However, in most scenarios there is a need to change at least some of the configuration parameters.

2.2.1 Setting a Profile

There are two high level profiles that can be used to optimize the sensor for different use cases. The default `ACC_SERVICE_ENVELOPE_PROFILE_MAXIMIZE_DEPTH_RESOLUTION` profile is recommended for high precision distance measurements at short distances. For other applications where the distance between the sensor and the measured object is larger than 0.25 m it is generally better to use the `ACC_SERVICE_ENVELOPE_PROFILE_MAXIMIZE_SNR` profile. In this profile the peaks in the signal response will be less distinct, but the sensor will transmit more energy per measurement which improves the signal to noise ratio.

Profile	Recommended Range	Max sweep length*
`ACC_SERVICE_ENVELOPE_PROFILE_MAXIMIZE_DEPTH_RESOLUTION`	0.06 m to 0.70 m	
`ACC_SERVICE_ENVELOPE_PROFILE_MAXIMIZE_SNR`	0.25 m to 2 m	

Use the function `acc_service_envelope_profile_set` to set the profile for the envelope service.

```
acc_service_envelope_profile_t profile = ACC_SERVICE_ENVELOPE_PROFILE_MAXIMIZE_SNR

acc_service_envelope_profile_set(acc_service_configuration, profile);
```

2.2.2 Running Average Factor (Advanced)

The running average factor is an optional setting that controls how the most recent sweep is weighed against previous sweeps. Valid range is between 0.0 and 1.0 where 0.0 means that no history is weighed in, i.e. filtering is effectively disabled. A factor of 0.99 means that the most recent sweep contributes with only 1%, and 99% of the signal value comes from previous sweeps.

The current default value for this setting is 0.7. When measuring objects in motion this value may be decreased. To improve SNR for static objects the running average factor could be increased to a value closer to 1.

```
// Optimize filter for objects in motion
acc_service_envelope_running_average_factor_set(envelope_configuration, 0.0);


// Optimize filter for static objects
acc_service_envelope_running_average_factor_set(envelope_configuration, 0.95);
```

2.3 Sweep Configuration

The sweep configuration parameters determine the sensor source and how the sweep data will be generated in the sensor. The sweep configuration parameters are common to all services and are therefore handled by a separate sweep configuration. Like other configuration parameters, the sweep parameters have reasonable default values, but in most applications, it is necessary to modify at least some them. To do this we must first obtain a sweep configuration handle.

```
acc_sweep_configuration_t sweep_configuration;

sweep_configuration = acc_service_get_sweep_configuration(envelope_configuration);

if (sweep_configuration == NULL) {
    /* Handle error */
}
```

2.3.1 Basic Configurations

Using the sweep configuration handle, we call access functions to set individual configuration parameters such as the sweep start and range.

```
// Set sweep start and length
acc_sweep_configuration_requested_range_set(sweep_configuration, .20, 0.4);
```

When using a connector board with multiple sensors the sensor id must also be set in the sensor configuration.

```
acc_sweep_configuration_sensor_set(sweep_configuration, 1);
```

2.3.2 Set Repetition Mode and Frequency

A repetition mode describes how the sensor behaves when producing data. There are currently two settable modes: Streaming and Max Frequency.

Streaming is the default mode and uses the sensor hardware as source for when to perform sweeps, i.e. the sensor hardware determines the timing. In this mode it is possible to set a desired frequency, see below.

```
// Set repetition mode streaming and the desired sweep frequency
acc_sweep_configuration_repetition_mode_streaming_set(sweep_configuration, 100);
```

The other repetition mode, Max Frequency, allows the host to perform sweeps continuously as fast as possible. The limiting factor on the update rate is now the whole system, i.e. both host and sensor. It is not possible to set a certain frequency in this mode. This mode does not guarantee any form of accurate timing. This mode is not compatible with Power Save Mode A (see chapter on Power Save Mode). Max frequency is set as below.

```
// Set repetition mode max frequency
acc_sweep_configuration_repetition_mode_max_frequency_set(sweep_configuration);
```

2.3.3 Set Sensor Gain (Advanced)

Advanced users can control the receiver gain in the sensor. This may be useful for example in cases when measuring reflections from strong reflectors close to the sensor. The gain value must be between 0.0 and 1.0, where 0.0 is the lowest gain and 1.0 is the highest gain.

```
// Decrease receiver gain
float current_gain;
current_gain = acc_sweep_configuration_receiver_gain_get(sweep_configuration);
acc_sweep_configuration_receiver_gain_set(sweep_configuration, 0.8 * current_gain);
```

2.3.4 Set Power Save Mode (Advanced)

Each power save mode corresponds to how much of the sensor hardware is shutdown between sweeps. Mode A means that the whole sensor is shutdown between sweeps while mode D means that the sensor is in its active state all the time. For each power save mode there will be a limit in the achievable update rate. Mode A will have the lowest update rate limit but also consumes the least amount of power for low update rates.

The update rate limits also depend on integration and range settings so for each scenario it is up to the user to find the best possible compromise between update rate, range and power consumption.

```
acc_sweep_configuration_power_save_mode_set
 (sweep_configuration, ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_B);
```

ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_A	Maximum power save
ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_B	High power save
ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_C	Limited power save
ACC_SWEEP_CONFIGURATION_POWER_SAVE_MODE_D	Sensor always active – no power save

3 Capturing Envelope Data

3.1 Creating and Activating the Service

After the envelope configuration has been prepared and populated with desired configuration parameters, the actual envelope service instance must be created. During the creation step all configuration parameters are validated and the resources needed by RSS are reserved. This means that if the creation step is successful we can be sure that it is possible to activate the service and get data from the sensor (unless there is some unexpected hardware error).

```
acc_service_handle_t envelope_handle = acc_service_create(envelope_configuration);

if (envelope_handle == NULL) {
    /* Handle error */
}
```

If the service handle returned from `acc_service_create` is equal to NULL, then some setting in the configuration made it impossible for the system to create the service. One common reason is that the requested sweep length is too long for the selected profile, but in general, looking for error messages in the log is the best way to find out why a service creation failed.

When the service has been created it is possible to get the actual number of samples (`data_length`) we will get for each sweep. This value can be useful when allocating buffers for storing the envelope data.

```
acc_service_envelope_metadata_t envelope_metadata;
acc_service_envelope_get_metadata(envelope_handle, &envelope_metadata);
uint16_t data_length = envelope_metadata.data_length;
```

It is now also possible to activate the service. This means that the radar sensor starts to do measurements.

```
acc_service_status_t service_status = acc_service_activate(envelope_handle);
```

3.2 Reading Envelope Data from the Sensor

The easiest way to read the envelope data from the sensor is to call the function `acc_service_envelope_get_next`. This function blocks until the next sweep arrives from the sensor and the envelope data is then copied to the `envelope_data` array.

```
uint16_t envelope_data[envelope_metadata.data_length];
acc_service_envelope_result_info_t result_info;


for (int i=0 ; i<50 ; i++)
{
    service_status = acc_service_envelope_get_next(envelope_handle,
        envelope_data,
        envelope_metadata.data_length,
        &result_info);

    if (service_status!= ACC_SERVICE_STATUS_OK) {
    /* Handle error */
    }

    /* Process Envelope Data */
}
```

3.3 Using Callback Functions to Receive Data (Advanced)

There is an optional way for the application to obtain envelope data from the API. Instead of doing multiple calls to the `acc_service_envelope_get_next` function, it is possible to register a callback function. The callback function is then automatically called by Radar System Services whenever there is new envelope data available from the sensor. The main advantage with using a callback function is that the main application is not blocked while waiting for the next sweep. The API overhead is also reduced slightly as a copy of the envelope data to a second internal buffer is avoided.

The callback function must be registered in the configuration step before creating the service. A pointer to a user defined data structure is also registered. This pointer will be passed to all calls to the register callback function. The user defined data structure should contain state information that needs to be saved between the calls to the callback function. It is also a good place for storing the length of the envelope data array.

```
typedef struct
{
     uint16_t     data_length;
     uint16_t     some_state_information;
} envelope_callback_user_data_t;

envelope_callback_user_data_t callback_user_data;


acc_service_envelope_envelope_callback_set(envelope_configuration,
                                           &envelope_callback,
                                           &callback_user_data);
```

Note that the code in the callback function will execute in a different thread than the rest of the application. This means that access to shared data structures or other shared resources needs to be synchronized properly to avoid race conditions. If the sensor is configured to run in repetition mode streaming with an explicit frequency set, you should avoid long running code in the callback function. If the callback function has not returned when the new data arrives from the sensor the data will be lost.

```
void envelope_callback(const acc_service_handle_t service_handle,
                       const uint16_t *envelope_data,
                       const acc_service_envelope_result_info_t *result_info,
                       void *user_reference)
{

     envelope_callback_user_data_t *callback_user_data = user_reference;

    /* Process Envelope Data */


}
```

4 Deactivating and Destroying the Envelope Service

Call the `acc_service_deactivate` function to stop measurements.

```
service_status = acc_service_deactivate(envelope_handle);

if (service_status!= ACC_SERVICE_STATUS_OK) {
              /* Handle error */
}
```

After the service has been deactivated it can be activated again to resume measurements or it can be destroyed to free up the resources associated with the service handle.

```
acc_service_destroy(&envelope_handle);
```

Finally, call `acc_rss_deactivate` when the application doesn't need to access the Radar System Services anymore. This releases any remaining resources allocated in RSS.

```
acc_rss_deactivate();
```

5 How to Interpret the Envelope Data

The envelope data should be interpreted as a single one-dimensional array, where each array element represents the amplitude of the reflected signal from objects at a particular radial distance from the sensor.

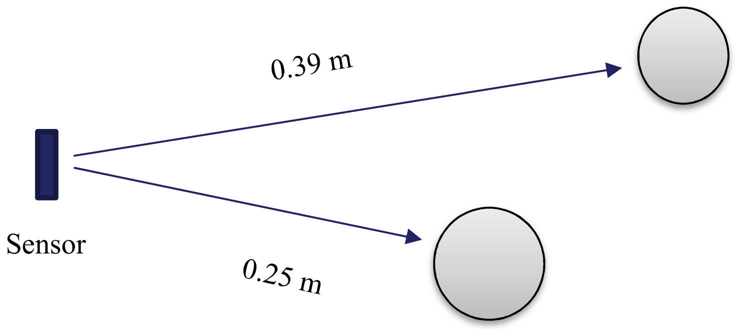
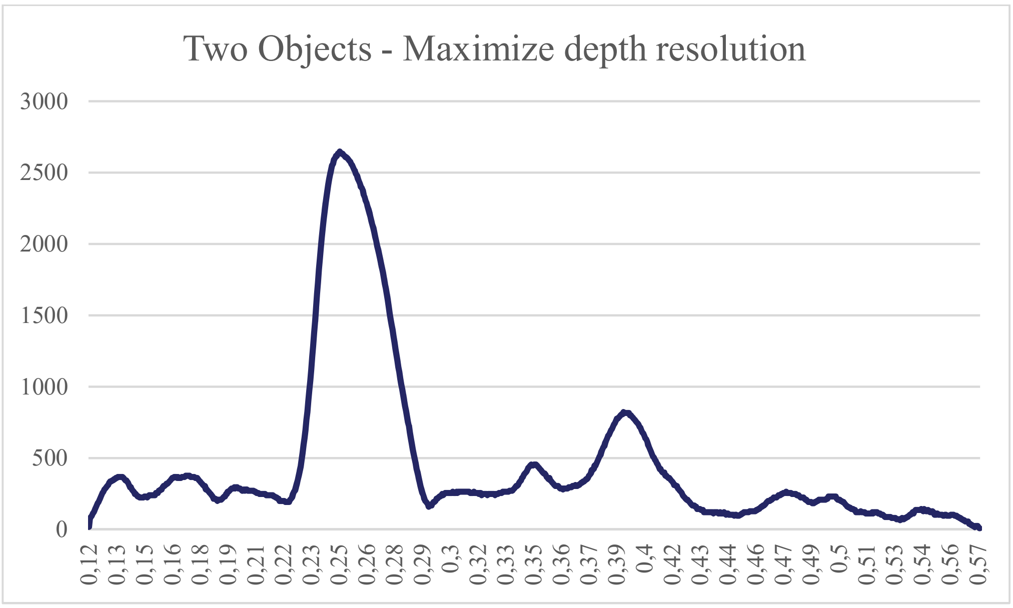
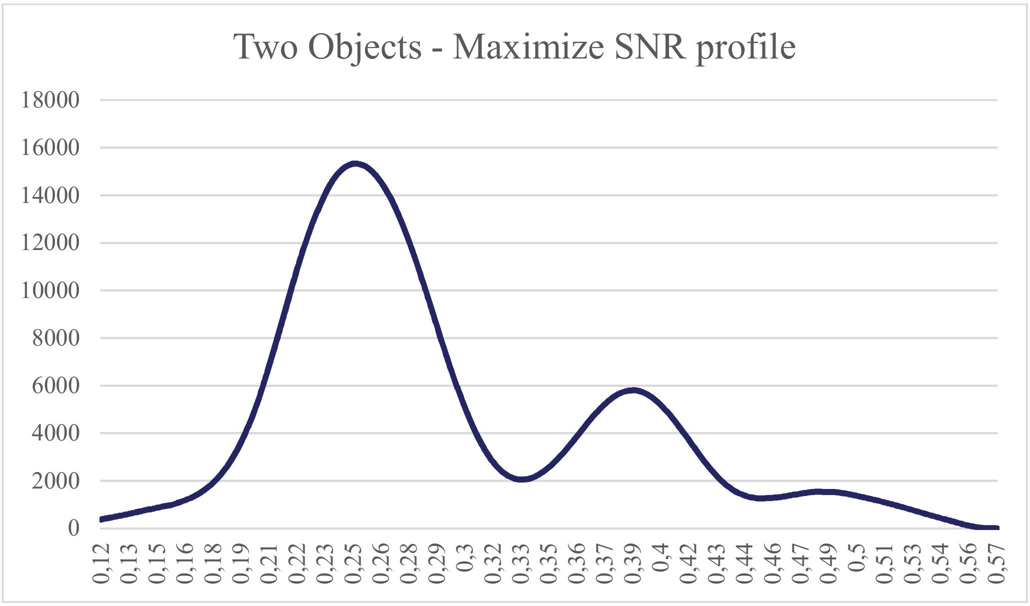


Figure 2- Measuring two objects

Assume that you have a setup with two objects like in the figure above. When plotting the envelope data recorded using the default maximum depth resolution profile you will get a graph similar to the one below.



When the Maximize SNR profile is used, the radar sends out more energy, and the receiver has a higher gain setting which leads to a higher amplitude in the received signal. The internal filter parameters are also different which gives a smoother signal where noise and fine details have been removed. The graph below shows the same two objects as before but with the Maximize SNR profile enabled.



5.1 Envelope Metadata

In addition to the array with envelope data samples, the metadata and the result info data structures provide side information that can be useful when interpreting the envelope data.

```
acc_service_envelope_metadata_t envelope_metadata;
acc_service_envelope_get_metadata(envelope_handle, &envelope_metadata);
```

The most important member variable in the meta data structure is `data_length` which holds the length of the envelope data array. The other member variables, `actual_start_m`, `actual_length_m` and `free_space_absolute_offset`, are needed in applications that needs accurate distance measurements.

metadata struct member	Explanation
`actual_start_m`	Actual start of the sweep - may differ from requested in the configuration.
`actual_length_m`	Actual length of the sweep - may differ from requested in the configuration.
`data_length`	Length of the envelope data array.
`free_space_absolute_offset`	Sensor specific offset error. Can be used to reduce the variation in distance measured from different sensors.

6 Implementing a Simple Distance Algorithm

6.1 Find Distance to Envelope Peak

In this example we will implement a simple distance algorithm that finds the distance to the strongest reflecting object by finding the highest peak in the envelope data. We start by writing a function to find the index for the value in the envelope data array with the highest amplitude.

```
int peak_index(uint16_t data[], uint16_t data_length) {
    uint16_t max = 0;
    int peak_idx = 0;

    for (int i = 0 ; i<data_length ; i++){
        if (data[i] > max ) {
            max = data[i];
            peak_idx = i;
        }
    }

    return peak_idx;
}
```

By using information on actual start and actual sweep length in the meta data structure we can transform the index to a distance. To avoid measuring the distance to some peak in the background noise when there is no object in front of the sensor we define an amplitude threshold that must be exceeded in order write out that we found a distance.

```
/* Process Envelope Data */

int peak_idx = peak_index(envelope_data, envelope_metadata.data_length);
unint16_t min_amplitude = 1000;

if (envelope_data[peak_idx] > min_amplitude) {
   float dist =  envelope_metadata.actual_start_m + peak_idx *
                 envelope_metadata.actual_length_m / (envelope_metadata.data_length - 1) -
                 envelope_metadata.free_space_absolute_offset;

   printf ("distance: %f, amplitude %d\n", dist, envelope_data[peak_idx]);
} else {
   printf ("amplitude under threshold\n");
}
```

Disclaimer