Lesson 18: SMS and Scheduled Function Result Callbacks

Created by Jacques Marais, last modified on Jul 17, 2017

> *As a System Admin I want to be notified via email about consecutive failures of SMS or e-mail messages.*

Lesson Outcomes

After this lesson you should:

- Have basic knowlegde of the `__sms_result_update__` and `__scheduled_function_result_update__` built-in functions
- Know how to use the **`@OnSmsResultUpdate`** and **`@OnScheduledFunctionResultUpdate`** functions to create callback functions for scheduled function and SMS result updates

Modified or Added App Files

`./model/objects/ScheduledFunctionFailureCounter.mez`

`./model/objects/SmsFailureCounter.mez`

`./services/ScheduledFunctionResultCallbacks.mez`

`./services/SmsResultCallbacks.mez`

`./web-app/lang/en.lang`

App Use Case Scenario

A typical feature for many Helium apps is to have scheduled functions or to send outbound SMS messages. These operations are processed asynchronously on the Helium backend. For various reasons, however, scheduled functions and SMS messages might fail. For this reason Helium provides developers with a mechanism to keep track of the status of scheduled functions and SMS messages. This is achieved by providing two built-in objects, meaning they are inherently available in each app without developers needing to include them in the app data model, and two annotations for callback functions. The feature described in this lesson includes functionality where these callbacks are used to record consecutive failures of scheduled functions and SMS messages and to then generate a notification by e-mail for the attention of system admin users.

Built-In Objects

The following built-in objects will be referenced in this lesson:

- `__sms_result_update__`
- `__scheduled_function_result_update__`

These objects are discussed in more detail in Lesson 19. For this lesson we will simply refer to the `success` attribute in both of these objects and the `doneProcessing` attribute which is exclusive to the `__sms_result_update__` function.

Annotations for Scheduled Function and SMS Result Update Callbacks

Firstly we need to add the following objects to our data model:

```
persistent object SmsFailureCounter {
    int consecutiveFailures;
    int totalFailures;
    int totalSuccesses;
    datetime lastUpdate;
}
```

```
persistent object ScheduledFunctionFailureCounter {
    int consecutiveFailures;
    int totalFailures;
    int totalSuccesses;
    datetime lastUpdate;
}
```

These objects will be used to record the total failures and consecutive failures of SMS messages and scheduled functions.

We now add two new units in presenter files under the services folder namely **`SmsResultCallbacks.mez`** and **`ScheduledFunctionResultCallbacks.mez`** respectively. We can now add our callback functions to these units:

```

 unit SmsResultCallbacks;

 @OnSmsResultUpdate
 void smsResultCallback(__sms_result__ smsResult) {

     // Get or create the counter even if we don't know
     SmsFailureCounter failureCounter = getOrCreateSmsF

     // If a failure has occurred, record it
     if(smsResult.success == false && smsResult.donePro
         failureCounter.consecutiveFailures = failureCo
         failureCounter.totalFailures = failureCounter.
         failureCounter.lastUpdate = Mez:now();
     }
     // Reset the consecutive failure count
     else if(smsResult.success == true && smsResult.don
         failureCounter.consecutiveFailures = 0;
         failureCounter.totalSuccesses = failureCounter
         failureCounter.lastUpdate = Mez:now();

```

Be sure to use the correct function signatures for your callback functions.

Keep in mind that only one callback function for SMS results updates is allowed. Similarly only one scheduled function result update callback function is allowed.

Helium performs multiple attempts to try and send SMS messages. If an SMS result update has a value for the `success` attribute of **`false`**, it is possible that the message will eventually be sent. It is therefore important to also check the `doneProcessing` attribute. If this is set to true, no more attempts to send the message will be made.

```
    }

    // Alert all system admins of consecutive failures
    if(failureCounter.consecutiveFailures >= 5) {
        notifySystemAdminsOfConsecutiveFailures(failur
    }
}
```

```
unit ScheduledFunctionResultCallbacks;


@OnScheduledFunctionResultUpdate
void scheduledFunctionResultCallback(__scheduled_funct

    // Get or create the counter even if we don't know
    ScheduledFunctionFailureCounter failureCounter = g

    // If a failure has occurred, record it
    if(scheduledFunctionResult.success == false) {
        failureCounter.consecutiveFailures = failureCo
        failureCounter.totalFailures = failureCounter.
        failureCounter.lastUpdate = Mez:now();
    }
    // Reset the consecutive failure count
    else {
        failureCounter.consecutiveFailures = 0;
        failureCounter.totalSuccesses = failureCounter
        failureCounter.lastUpdate = Mez:now();
    }

    // Alert all system admins of consecutive failures
    if(failureCounter.consecutiveFailures >= 5) {
        notifySystemAdminsOfConsecutiveFailures(failur
    }
}
```

Note the use of the following:

- **`@OnSmsResultUpdate`** and **`@OnScheduledFunctionResultUpdate`** annotations
- `__sms_result__` and `__scheduled_function_result__` functions parameters
- `success` and `doneProcessing` attributes

Lesson Source Code

Lesson 18.zip

No labels