Application Integration Registrar (AIR)

Developer guide

By
Mohamed Roshan

Contents

Application Integration Registrar (AIR) developer guide.

1. Introduction to AIR

Application Integration Registrar or AIR, simply is an access control system to other Ekwa internal applications. There are some principle similarities to Ekwa UAACS. But the difference is AIR only controls communications and related access between applications.

What this means is if an application, for example app A wants to communicate with app B in order for it to carry out a task on behalf of app B. Under normal conditions such an event can be carried out only if both apps A and B are built with code to carry out requests made by each other. The developers of both apps need to communicate with each other and design and develop the required modifications to the apps. Hence each integration requirement is built on a case by case basis at the point of requirement.

Now Suppose if app A needs to communicate with another app, say app C, developers will have to edit the code of both apps in order to make a connection between the two.

Needless to say this kind of linking between two or more applications is time consuming, not flexible and above all this is prone for errors which are hard to debug.

With the implementation of AIR, developer has to do nothing but to create a connection between two apps using the AIR application interface and then to use a specific library which handles the required communication between the two apps that need the integration. In this each integration between two or more apps are properly recorded and registered in the AIR app. In other words no app can access or execute code of other apps without getting through the access control of AIR. The built-in library will ensure the verification of authenticity of the integration request as well as log all access and executions for future verification if the need be.

All request are defined with outcome status indicators and they are well streamlined hence a developer only has to add 2 -3 lines of pre-defined codes wherever appropriate

This approach is much more flexible, secure, easy to maintain and have more control over the integrations.

2. Concept of AIR

The basic concept is that AIR will act like a middleman in between the communication of two applications that needs to be integrated.

A communication between two apps can happen only if the connection are defined in AIR.

At the point of defining the connection AIR needs detailed information about both the apps to be integrated. Hence such details relating to the apps are defined by the app developer in a predefined XML feed format. For example functions of the app that could be used/accessed by other apps for integration and their parameters, URL to access the app function,function method etc.

Once those data are given in AIR, developer has to use a specific library to initiate the communication.

If we were to study the example given above,

Developer first has to define an integration for apps A & B(let's name it "A -> B") in AIR where the app A is the calling/requesting application and app B is the called/receiving application.

Now whenever app A wants to communicate with app B, app A has to obtain a token from AIR for the defined integration A -> B.

To do this app A needs to incorporate and uses the library developed for this propose.

Once AIR received a request from app A for a token, it checks whether any information contains about app A in AIR database.

Since app A has already registered an integration, AIR proceeds and creates a token and sends it to app A along with other application's information such as app B's function URL, parameters, etc.

Upon obtaining the token and the app B's function URL and parameters information, app A now can call app B's required function directly with the token.

App B on the other end receives app A's call and the token.

App B first checks the URL request to see whether it's an internal app execution call or an external AIR related call.

Once app B determined this call is from an external source, it then proceed to verify the token from AIR with the help of AIR related library.

AIR receives token verification call from app B and takes appropriate steps to verify the token and responds back to app B with token validity.

Upon receiving token validity response, app B either execute app A's call or denies to do so. App B also receives data required for the app B’s login environment settings in the same response.

This is how the communication take place between applications integrations when implemented through AIR.

As intended it is very easy to implement though AIR does a load of work such as UAACS checks associated with user who is using the app and all sort of other check ups behind the seen.

3. What are the main components?

There are 4 main components involved when implementing AIR for app integrations.

I. **AIR application**
AIR application itself.

II. **AI_Utilities.php lib**
AI_Utilities.php library is what resides inside the application which intends to communicate with other apps.
This library act as a slave component for AIR master.
Token requesting, token verification, processing responses from AIR is happening in this component.

III. **Calling application**
This is the application which makes the call to execute a function in other application (called application)

IV. **Called application**
This is the application which receives the call from calling application.

4. How to implement?

To implement AIR, follow the exact steps described below in exact order.

setups to be done in both calling app and called app. Before making a connection in AIR, Called app has to be set up first. This is a must

1. Identify called app's required function/method and add code segments in appropriate locations. All those code segments are to be given and explained in following sections.
2. Create the share_function.xml file and put it on root folder of the **called** app.
3. Make sure *share_function.xml* of **calling** app is existing and make sure its residing on root folder of **calling** app. If not ask calling app developer to create one.
4. Add and load AR_Utilities.php class for both apps.
5. Define the integration in AIR. This process explained in step by step below. Ensure you have the generated *Integration Id* ready when asked for.
6. Add code segments on calling app method in appropriate locations.
7. Handle error/status codes where appropriate as explained in foregoing steps.
8. Test and debug

Pre-requisitions

1. Both calling and called applications must have UAACS access controlling enabled.
2. Both calling and called apps must be equipped with share_function.xml feed. To learn how to create one please refer 'How to create share_function.xml feed' section.
3. Called app's method must have proper error handling. To learn how to do this please refer 'Error handling code setup' section.

4. Define **AR_URL** constant variable at config/constants.php file if your application is a CI orelse define it in your application where it is appropriate.

   You need to give **AIR** application test URL as follows, 'http://bizydads.com/appqa/AI/AI/index.php/appregistrar/' and live URL As 'http://kindersigns.org/AIR/index.php/appregistrar/'
   (Note: do not forget to add **'/'** at the end of the URL.)

5 Steps to setup a function as called app function (called function)

These steps should be done by called application developer (If the application is generated through new CIGEN, then this step is already done for you). Once a function is identified as an API function or shared function, developer should follow each step given below,

1. Include AR_Utilities.php class in your CI app library folder if it's a CI app. If the app is not CI based you can include it as a general include.
2. Declare following private properties in the **Called Application**'s controller class if it is a CI app. Otherwise declare as global variables.

```
//AIR related vars
private $AIR_call_flag = false; // Use this flag to select appropriate output
private $debug =  true;
private $AR_Utilities = null;
```

3. After that add following code segment inside the constructor. This will create an object which activates called app functions of AI_Utilities class.

```
// Load AIR Lib to check and configure settings/views if its an AIR related call
$this->load->library('AR_Utilities');

//Create AIR obj to act as    called app
$this->AR_Utilities = new AR_Utilities("called_app");
if($this->AR_Utilities->validityStatus['validity'] == 'VALID'){
      $this->AIR_call_flag = true;
}
```

The purpose of using `$this->AIR_call_flag` is to select which output to be executed. If this flag is set to TRUE which means it is an external call so handle it accordingly. If it is FALSE then proceed with normal execution.

validityStatus is an array property of AR_Utilities library. Which returns '**validity**' key as **'VALID'** or **'INVALID'** string value.

If the key **validity of validityStatus** property returns **'INVALID'** then it will additionally returns **'errorCode'** and **'error'** keys with relevant data.

In some occasions when you need to output errors or messages you can use AR_Utilities **header_output()** method.

header_output() is a AR_Utility method which accepts 2 parameters. First one is a code. Which should be an integer value. The second one is a string value which can be a string.

Ex:

```
$this->AR_Utilities->header_output(
                                $this->validityStatus['errorCode'],
                                $this->validityStatus['error']);
```

You can also send your own custom error messages and codes as follows,

```
$this->AR_Utilities->header_output(511,'Token was not given!');
```

You should always use **header_output()** method If you want to return any messages to calling app.

Ex:

```
if(empty($actual_start_date)){
       $this->AR_Utilities->header_output(520,'Actual Start Date cannot be empty');
       exit;
}else if(empty($actual_end_date)){
       $this->AR_Utilities->header_output(521,'Actual End Date cannot be empty');
       exit;
}
```

6 Calling application setup for calling app developer

1. Register an integration with AIR as follows,

    1.1. Click on 'Add' button on AIR toolbar to load **'Add New Application Integration'** form
    1.2. Give a name to identify the integration for **'Integration Name'** text box
    1.3. Select the calling application from **'Calling Application'** drop down menu
    1.4. Give the path of calling application's share_functions.xml feed for **'Calling App XML URL'** text box
    1.5. As soon as you specify correct URL of **'share_function.xml'** and take out the courser from the text box, AIR creates a drop down menu with specified functions from the XML file and shows at **'Calling Application Function '**
    1.6. Select appropriate function from **'Calling Application Function'**
    1.7. Select **'Called application'**
    1.8. Give called application's share_function.xml URL. This would be usually application root path.
    1.9. Select application's called function from the drop down. This would appear as soon as you specify the path of XML feed
    1.10. Finally you have to give specific values for 'Function parameters and query string data variables' section. This may different for each application requirements. Called application's share_function.xml feed will contain instructions on how to fill these section.
    1.11. Press 'Save' button to save the connection.

    1.12. Once the connection is added in AIR you would see it in AIR grid. Integration Id is what you will need to remember for other steps.

2. In your calling application add AI_Utilities.php file in library folder if your app is built with CodeIgnitor. Or else you can have this as a general *include* class
3. In your application, the function which needs to access the other application's function **(called app function)** is known as calling function. Inside the calling function, load AR_Utilities library and create an instance.

    If you have non CI application make the AR_Utilities var as global.

```
$this->load->library('AR_Utilities');
// Note AR_Utility does not accept any parameter for calling app instantiation
$this->AR_Utilities = new AR_Utilities();
```

4. After that assign values to AR_Utilities object's properties as follows,

```
$this->AR_Utilities->ai_id = <Your Integration Id from AIR grid> ;
$this->AR_Utilities->calling_app_user_id = $_SESSION['user_id'];
$this->AR_Utilities->calling_app_id = <your app id from UAACS feed>;
$this->AR_Utilities->calling_app_function_id = < calling app function id>;
$this->AR_Utilities->called_app_id = <Id of called app >;
$this->AR_Utilities->debug = <to see debug info true or false>;
```

5. Add codes to obtain token information

```
$token_data = $this->AR_Utilities->request_token($this->AR_Utilities->debug);
```

6. Additionally if in case AIR returns an error, it will return as an array which has 2 array elements. 1 ErrorCode, 2 Error. To handle the errors add error handling code as shown below,

```
if(isset($token_data['ErrorCode'])){
	echo 'Error: '.$this->token_data['ErrorCode'].' '.$this->token_data['Error'];
	exit;
}
```

7. Store the token data in a session

```
$session['token_data'] = $token_data;
```

8. Now add following execution code to get called application output,

```
$output = $this->AR_Utilities->init_call(
                                        true,
                                        $this->AR_Utilities->debug,
                                        $_SESSION['token_data']['token']);
```

init_call() method in AR_Utilities class accepts 3 parameters. To get the complete output such as headers/XML/HTML from called application set first parameter as **TRUE**. Set it as **FALSE** to get header messages only. No HTML/XML content will be returned.

Second parameter is for debug information. If it is set to **TRUE** it will output debug information. By default it is set to False.

Third parameter is to accept token. If you supply a valid token init_call will return outputs. If the token is not valid it will return error messages.

9. If you were to pass parameter values or query string values to called app,you should make it as **$GLOBALS** variable. **Note that all var names are expected to be in UPPERCASE here.**

10. For example if app B (called application) requires CID , TYPE_ID values as parameters then you need to specify them as follows,

```
// make CID,TYPE_ID global as follows
$GLOBALS['CID'] = 20;
$GLOBALS['TYPE_ID'] = 345;

//then call init_call();
$output = $this->AR_Utilities->init_call(false,$this->AR_Utilities->debug);
```

init_call() method is intelligent enough to assign appropriate parameters and query strings inside its method. Developer has to just call it after making variables global.

11. After execution $output will contain called application's output. You can then process it as per your need.

Tip: use var_dump($output) to see what it contains before coding further.

Now that you have set up both called app and calling app functions, it is time to test the implementation of AIR.

When you call calling app function you should see desired output from called application method. If you, var dump **$output** inside calling app method you will see array elements along with results.

Try testing by giving invalid token or other data to see if you get error messages correctly.

7 How to handle multiple calling app functions in a calling application

There can be few instances that a calling application can have multiple calling app functions. In a situation like this you can easily overcome this problem by using multiple session keys to store requested tokens and then implement them with appropriate calling function.

For example,

Say in a calling application, there are 2 calling functions namely function a() function b(). Both needs to obtain token from AIR in order to execute called application functions.

To do this we first create 2 entries in AIR and keep both ids ready to use.

In function a() body we do code as follows.

```
$this->load->library('AR_Utilities');
$this->AR_Utilities_a = new AR_Utilities();

$this->AR_Utilities_a->ai_id = <Your Integration Id from AIR grid> ;
$this->AR_Utilities_a->calling_app_user_id = $_SESSION['user_id'];
$this->AR_Utilities_a->calling_app_id = <your app id from UAACS feed>;
$this->AR_Utilities_a->calling_app_function_id = < calling app function id>;
$this->AR_Utilities_a->called_app_id = <Id of called app >;
$this->AR_Utilities_a->debug = <to see debug info true or false>;

$token_data_a = $this->AR_Utilities_a->request_token($this->AR_Utilities->debug);
$_SESSION['token_data_a'] = $token_data_a;


if(isset($token_data_a['ErrorCode'])){
       echo 'Error: '.$token_data_a['ErrorCode'].' '.$token_data_a['Error'];
       exit;
}
$output = $this->AR_Utilities_a->init_call(
                                        true,
                                        $this->AR_Utilities_a->debug,
                                        $_SESSION['token_data_a']['token']);
var_dump($output);
```

Similarly, in function b() body we do the same code and instead of `$token_data_a` we provide a different name for the AIR object.

The main objective is to store sessions in different session ids and use them appropriately when executing **init_call()**

8 Troubleshooting

In case if you could not see expected output you can troubleshoot by following steps.

1. Go through this documentation again and double check if the steps are correctly implemented
2. Set AI_Utiity class's $debug property to true and see the output to understand error messages
3. You can then get AIR application's direct URL to check how token issue/ token verification function's outputs. From that you can see what the error is about.
4. Check your log files to see error messages. If you have not implemented log object then you should still be able to see it in CI log or in system log
5. Get your AI Id and open up AIR Edit form to see the values, XML feed URLs and check their data.
6. Check if you get any error codes as output if so then cross check with given error codes in this documentation. Then make necessary changes.

9 Error handling code setup

When it comes to error handling in AIR implemented applications, we have to receive header messages from called app and we need a mechanism to capture those header messages at calling app end.

AR_Utility class is already equipped to capture header messages and process it

When calling app calles **init_call()** method it gets the output from called app if there is any. Otherwise it gets header messages from called app.

Those header messages then be processed and formatted to present to the user.

So you do not need to handle header messages manually for calling app set up.

But you need to set up header messages for called app function.

Under **Steps to setup a function as called app function (called function)** section we have seen how to set up called app function to perform when a AIR related call is received.

Point no 3 of that section explains how to set up validation error handling as header messages.

For example following code segment will validate for empty values for actual start date and actual end date and if they are empty header functions are used to return error messages,

```
if(empty($actual_start_date)){
	$this->AR_Utilities->header_output(520,'Actual Start Date cannot be empty');
	exit;
}else if(empty($actual_end_date)){
	$this->AR_Utilities->header_output(521,'Actual End Date cannot be empty');
	exit;
}
```

Similar concept can be used to handle error or notification for your application as well.

10 AIR Error codes

515 - Error: UAACS data feed error! - This error occurs when there is a problem related to UAACS feed data. This can happen if calling/called app ids are not valid or supplied share_function XML feed has some errors.

514 - Error: AIR could not fetch details of given calling app id, called app id and calling app function id! - This error occurs when supplied calling app id, called app id, and calling app id values are not correct

513 - Error: You do not have access rights to called application! - This error occurs when the calling app user has no rights over called application and trying to access it.

512 - Error: Unauthorized called application call! - This error occurs when invalid called application is trying to use AIR

511 - Error: Unauthorized calling application call! - This error occurs when invalid calling application is trying to use AIR

510 - Error: Unauthorized server access! - This error occurs when unauthorized server is trying to access AIR

Error codes of token verification processing

526 - Error: You do not have access permission to called application! - This error occurs when user has no permission to use called application

525 - Error: No permission! - This error occurs when user has no permission over called application

524 - Error: Non existing token! - This error occurs when supplied token is invalid

523 - Error: Unauthorized host! - This error occurs when calling user host is unknown

522 - Error: Invalid called app! - This error occurs when supplied called application given is incorrect

521 - Error: Invalid calling app! - This error occurs when supplied calling application given is incorrect

520 - Error: Token was not provided! - This error occurs when the token is not provided for verification

11 About function get_related_ar_links($calling_app_id=0, $calling_app_function_id=0, $debug=false)

There can be situations that you may need to fetch all AIR linkages which is created for a particular calling application. So that you can perform additional tasks or can create a chain of tasks in one go. This can be achieved by using this function.

This function will return an associative array upon successful retrieval of all related links.

12 How to create share_function.xml feed

A basic share_function.xml should look like this.

```
<api>
<app_name>(Your application name)</app_name>
<app_id>(Your app UAACS id)</app_id>
<functions>
        <function  id="( your function id)"  name="(your function name)">
        <roles>(role, ids, must, be, comma, separated)</roles>
        <parameters>
                <parameter>
                        <display_name>(Your function parameter name)</display_name>
                        <data_type>(Parameter data type)</data_type>
                        <help_text>
                                (Provide description about this parameter. These help texts
                                to be shown on AIR integration form. This description should
                        be informative)
                        </help_text>
                </parameter>
        </parameters>

        <url>(Your function URL as it appears in form action or as a get URL)</url>
        <url_method>(URL method either get or post)</url_method>

        <query_string_vars>

                <(Your querystring var name)>
                        <display_name>(query string para name)</display_name>
                        <data_type>(query string para data type)</data_type>

                        <help_text>
                        (Similar to parameter help text. Explain what this query string parameter
does etc.)
                        </help_text>

                        <default>(Provide any default values. Use '@@variable_name@@'if you want
to make it dynamic. For more info see below)</default>
                <editability>(true/false)</editability>
                </(Your querystring var name)>

                <(Your querystring var name)>
                        <display_name>(query string para name 2)</display_name>
                        <data_type>(query string para name 2 data type)</data_type>

                        <help_text>( help text about query string para name 3)</help_text>

                        <default>( provide if any default values)</default>
                        <editability>(true/false)</editability>
                        </(Your querystring var name)>
```

```
        </query_string_vars>
        </function>
</functions>
</api>
```