**User Hooks in Oracle HRMS**

### User Hook Concepts

There are many times where we need put some extra logic before or after happening of some business event. In Such cases, we use user hook API. It is a functionality provided in Oracle HRMS through which you can have more control on application w.r.t implementing business rules.

**How it Works:**

In Oracle HRMS, Oracle has provided location in HRMS APIs, where customer can put his business logic. When API processing reaches a user hook, core product processing stops and any customer specific logic for that event is executed. After processing of customer specific logic, main API resumes its processing. These are normally used in scenarios where we want to put extra logic to add functionality not supplied directly by Oracle. This could include business events like

**1.** Validate Data is EIT before or after insertion either through self-service or core HR  
**2.** Validate Data is SIT before or after insertion either through self-service or core HR  
**3.** Validating particular customer data: For example, you could limit grade step promotions to a maximum of one step.  
**4.** Maintaining additional data in your own user defined tables  
**5.** Detecting that a particular business event has occurred: If the event was an employee termination process, for example, this could be made to send a message to your security database disabling the employee’s security pass.

**Attention:** You should not manually edit the API source code files supplied by Oracle If you do modify these codes, Oracle Applications will be unable to support the product, and upgrades may not be possible. Oracle Applications only supports direct calls to the published APIs. Direct calls to any other server-side package procedures or functions that are written as part of the Oracle HRMS Applications suite are not supported.

**Note:** All Oracle HRMS forms does not use HRMS APIs, User hook can be implemented for only those forms which perform functions through API. The HR\_API\_HOOKS table holds all available API hooks for APIs in HR\_API\_MODULES table.

### Implementation Steps for User Hooks

Four steps are required to implement API User Hooks:

**1.** Identifying the correct User Hook API where you want to put your custom logic  
**2.** Create PL/SQL procedure to execute your logic.  
**3.** Register your procedure with one or more specific user hooks.  
**4.** Run the pre-processor program that hooks your PL/SQL procedure to the hook(s).

Let’s Assume we want to put a logic to stop a user if he applied annual leave more than 30 days. It should validate before Absence Creation. So we have to perform following steps in sequence

**Step 1 – Identifying the correct User Hook API**

SELECT ahk.api\_hook\_id,

ahk.api\_module\_id,

ahk.hook\_package,

ahk.hook\_procedure,ahm.module\_name

FROM hr\_api\_hooks ahk, hr\_api\_modules ahm

WHERE ahm.module\_name like '%CREATE\_SIT%'

AND ahm.api\_module\_type = 'BP'

AND ahk.api\_hook\_type = 'AP'

AND ahk.api\_module\_id = ahm.api\_module\_id

It will show all Person Absence relevant APIs. We need to select the correct API that matched with our requirement. For Example for above mentioned requirement, we shall select CREATE\_PERSON\_ABSENCE\_A API. We shall note its API\_HOOK\_ID (2879) and API\_MODULE\_ID (1279) as API\_HOOK\_ID will be used at the time of registration of user hook mentioned in Step 3 ahead and API\_MODULE\_ID will be needed in running the processor in Step 4.

My custom Package, package name, PROCEDURE name is not dependent.

CREATE OR REPLACE PACKAGE TGC\_USERHOOK\_SIT\_PG AS

*/\**

*--QUERY FOR FIND CUSTOM PROCEDURE CODE THAT IS EXISTS IN HOOK CALL TABLE*

*SELECT \**

*FROM HR\_API\_HOOK\_CALLS*

*WHERE api\_hook\_id = 3840*

*AND API\_HOOK\_CALL\_ID=1281;*

*\*/*

PROCEDURE TGC\_EMPLOYEE\_SIT\_VALIDATION(P\_PERSON\_ID in NUMBER,

P\_BUSINESS\_GROUP\_ID in NUMBER,

P\_ID\_FLEX\_NUM in NUMBER,

P\_EFFECTIVE\_DATE in DATE,

P\_COMMENTS in VARCHAR2,

P\_DATE\_FROM in DATE,

P\_DATE\_TO in DATE,

P\_REQUEST\_ID in NUMBER,

P\_PROGRAM\_APPLICATION\_ID in NUMBER,

P\_PROGRAM\_ID in NUMBER,

P\_PROGRAM\_UPDATE\_DATE in DATE,

P\_ATTRIBUTE\_CATEGORY in VARCHAR2,

P\_ATTRIBUTE1 in VARCHAR2,

P\_ATTRIBUTE2 in VARCHAR2,

P\_ATTRIBUTE3 in VARCHAR2,

P\_ATTRIBUTE4 in VARCHAR2,

P\_ATTRIBUTE5 in VARCHAR2,

P\_ATTRIBUTE6 in VARCHAR2,

P\_ATTRIBUTE7 in VARCHAR2,

P\_ATTRIBUTE8 in VARCHAR2,

P\_ATTRIBUTE9 in VARCHAR2,

P\_ATTRIBUTE10 in VARCHAR2,

P\_ATTRIBUTE11 in VARCHAR2,

P\_ATTRIBUTE12 in VARCHAR2,

P\_ATTRIBUTE13 in VARCHAR2,

P\_ATTRIBUTE14 in VARCHAR2,

P\_ATTRIBUTE15 in VARCHAR2,

P\_ATTRIBUTE16 in VARCHAR2,

P\_ATTRIBUTE17 in VARCHAR2,

P\_ATTRIBUTE18 in VARCHAR2,

P\_ATTRIBUTE19 in VARCHAR2,

P\_ATTRIBUTE20 in VARCHAR2,

P\_SEGMENT1 in VARCHAR2,

P\_SEGMENT2 in VARCHAR2,

P\_SEGMENT3 in VARCHAR2,

P\_SEGMENT4 in VARCHAR2,

P\_SEGMENT5 in VARCHAR2,

P\_SEGMENT6 in VARCHAR2,

P\_SEGMENT7 in VARCHAR2,

P\_SEGMENT8 in VARCHAR2,

P\_SEGMENT9 in VARCHAR2,

P\_SEGMENT10 in VARCHAR2,

P\_SEGMENT11 in VARCHAR2,

P\_SEGMENT12 in VARCHAR2,

P\_SEGMENT13 in VARCHAR2,

P\_SEGMENT14 in VARCHAR2,

P\_SEGMENT15 in VARCHAR2,

P\_SEGMENT16 in VARCHAR2,

P\_SEGMENT17 in VARCHAR2,

P\_SEGMENT18 in VARCHAR2,

P\_SEGMENT19 in VARCHAR2,

P\_SEGMENT20 in VARCHAR2,

P\_SEGMENT21 in VARCHAR2,

P\_SEGMENT22 in VARCHAR2,

P\_SEGMENT23 in VARCHAR2,

P\_SEGMENT24 in VARCHAR2,

P\_SEGMENT25 in VARCHAR2,

P\_SEGMENT26 in VARCHAR2,

P\_SEGMENT27 in VARCHAR2,

P\_SEGMENT28 in VARCHAR2,

P\_SEGMENT29 in VARCHAR2,

P\_SEGMENT30 in VARCHAR2,

P\_CONCAT\_SEGMENTS in VARCHAR2,

P\_ANALYSIS\_CRITERIA\_ID in NUMBER,

P\_PERSON\_ANALYSIS\_ID in NUMBER,

P\_PEA\_OBJECT\_VERSION\_NUMBER in NUMBER);

END TGC\_USERHOOK\_SIT\_PG;

CREATE OR REPLACE PACKAGE BODY TGC\_USERHOOK\_SIT\_PG AS

PROCEDURE TGC\_EMPLOYEE\_SIT\_VALIDATION(P\_PERSON\_ID in NUMBER,

P\_BUSINESS\_GROUP\_ID in NUMBER,

P\_ID\_FLEX\_NUM in NUMBER,

P\_EFFECTIVE\_DATE in DATE,

P\_COMMENTS in VARCHAR2,

P\_DATE\_FROM in DATE,

P\_DATE\_TO in DATE,

P\_REQUEST\_ID in NUMBER,

P\_PROGRAM\_APPLICATION\_ID in NUMBER,

P\_PROGRAM\_ID in NUMBER,

P\_PROGRAM\_UPDATE\_DATE in DATE,

P\_ATTRIBUTE\_CATEGORY in VARCHAR2,

P\_ATTRIBUTE1 in VARCHAR2,

P\_ATTRIBUTE2 in VARCHAR2,

P\_ATTRIBUTE3 in VARCHAR2,

P\_ATTRIBUTE4 in VARCHAR2,

P\_ATTRIBUTE5 in VARCHAR2,

P\_ATTRIBUTE6 in VARCHAR2,

P\_ATTRIBUTE7 in VARCHAR2,

P\_ATTRIBUTE8 in VARCHAR2,

P\_ATTRIBUTE9 in VARCHAR2,

P\_ATTRIBUTE10 in VARCHAR2,

P\_ATTRIBUTE11 in VARCHAR2,

P\_ATTRIBUTE12 in VARCHAR2,

P\_ATTRIBUTE13 in VARCHAR2,

P\_ATTRIBUTE14 in VARCHAR2,

P\_ATTRIBUTE15 in VARCHAR2,

P\_ATTRIBUTE16 in VARCHAR2,

P\_ATTRIBUTE17 in VARCHAR2,

P\_ATTRIBUTE18 in VARCHAR2,

P\_ATTRIBUTE19 in VARCHAR2,

P\_ATTRIBUTE20 in VARCHAR2,

P\_SEGMENT1 in VARCHAR2,

P\_SEGMENT2 in VARCHAR2,

P\_SEGMENT3 in VARCHAR2,

P\_SEGMENT4 in VARCHAR2,

P\_SEGMENT5 in VARCHAR2,

P\_SEGMENT6 in VARCHAR2,

P\_SEGMENT7 in VARCHAR2,

P\_SEGMENT8 in VARCHAR2,

P\_SEGMENT9 in VARCHAR2,

P\_SEGMENT10 in VARCHAR2,

P\_SEGMENT11 in VARCHAR2,

P\_SEGMENT12 in VARCHAR2,

P\_SEGMENT13 in VARCHAR2,

P\_SEGMENT14 in VARCHAR2,

P\_SEGMENT15 in VARCHAR2,

P\_SEGMENT16 in VARCHAR2,

P\_SEGMENT17 in VARCHAR2,

P\_SEGMENT18 in VARCHAR2,

P\_SEGMENT19 in VARCHAR2,

P\_SEGMENT20 in VARCHAR2,

P\_SEGMENT21 in VARCHAR2,

P\_SEGMENT22 in VARCHAR2,

P\_SEGMENT23 in VARCHAR2,

P\_SEGMENT24 in VARCHAR2,

P\_SEGMENT25 in VARCHAR2,

P\_SEGMENT26 in VARCHAR2,

P\_SEGMENT27 in VARCHAR2,

P\_SEGMENT28 in VARCHAR2,

P\_SEGMENT29 in VARCHAR2,

P\_SEGMENT30 in VARCHAR2,

P\_CONCAT\_SEGMENTS in VARCHAR2,

P\_ANALYSIS\_CRITERIA\_ID in NUMBER,

P\_PERSON\_ANALYSIS\_ID in NUMBER,

P\_PEA\_OBJECT\_VERSION\_NUMBER in NUMBER)

IS

BEGIN

HR\_UTILITY.SET\_MESSAGE(0, 'HELLO');

HR\_UTILITY.RAISE\_ERROR;

END TGC\_EMPLOYEE\_SIT\_VALIDATION;

END TGC\_USERHOOK\_SIT\_PG;

**3. Register your procedure with one or more specific user hooks.**

We shall use the API\_HOOK\_ID identified in Step 1 in the parameter p\_api\_hook\_id. Through this API, custom logic will be registered against user hook.

DECLARE

L\_API\_HOOK\_ID NUMBER:= 2879; --🡺 HOOK IP WHERE YOU HAVE TO ADD UR CODE

L\_API\_HOOK\_CALL\_ID NUMBER; ---OUT PARAMETER

L\_OBJECT\_VERSION\_NUMBER NUMBER; --- OUT PARAMETER

L\_SEQUENCE NUMBER;

BEGIN

SELECT HR\_API\_HOOKS\_S.NEXTVAL

INTO L\_SEQUENCE

FROM DUAL;

HR\_API\_HOOK\_CALL\_API.CREATE\_API\_HOOK\_CALL

(FALSE, --> VALIDATION

TO\_DATE('01-JAN-1952','DD-MON-YYYY'), => P\_EFFECTIVE\_DATE

L\_API\_HOOK\_ID,

'PP', --> P\_API\_HOOK\_CALL\_TYPE

L\_SEQUENCE, ---HOOK ID SEQUENCE

'Y', --- ENABLED\_FLAG

'TGC\_USERHOOK\_PKG',---PACKAGE NAME CUSTOM

'TGC\_ABS\_LEAVE\_VALIDATIONS',---PROCEDURE NAME CUSTOM

L\_API\_HOOK\_CALL\_ID, ---OUT PARAMETER

L\_OBJECT\_VERSION\_NUMBER ---OUT PARAMETER

);

DBMS\_OUTPUT.PUT\_LINE('L\_API\_HOOK\_CALL\_ID'|| L\_API\_HOOK\_CALL\_ID);

END ;

Related Information Of HR\_API\_HOOK\_CALL\_API.CREATE\_API\_HOOK\_CALL

/\*

HR\_API\_HOOK\_CALL\_API.CREATE\_API\_HOOK\_CALL

(P\_VALIDATE => FALSE,  
P\_EFFECTIVE\_DATE => TO\_DATE(’01-JAN-1952′,’DD-MON-YYYY’),  
P\_API\_HOOK\_ID =>L\_API\_HOOK\_ID NUMBER,  
P\_API\_HOOK\_CALL\_TYPE => ‘PP’,  
P\_SEQUENCE => L\_SEQUENCE,  
P\_ENABLED\_FLAG => ‘Y’,  
P\_CALL\_PACKAGE => ‘LSG\_USERHOOK\_PKG’, — CUSTOM PACKAGE  
P\_CALL\_PROCEDURE => ‘LSG\_CREATE\_ABS\_VAL\_P’, — CUSTOM PROCEDURE  
P\_API\_HOOK\_CALL\_ID => L\_API\_HOOK\_CALL\_ID,  
P\_OBJECT\_VERSION\_NUMBER => L\_OBJECT\_VERSION\_NUMBER);

\*/

After this step, you shall be able to see the reference of your custom package/ procedure in table HR\_API\_HOOK\_CALLS. You can check it by using following query

SELECT \*  
FROM HR\_API\_HOOK\_CALLS  
WHERE api\_hook\_id = 2879;

**4. Run the pre-processor program**

Run pre-processor script hrahkone.sql with module name as parameter (PER\_TOP/admin/sql/hrahkone.sql).It compile API hook. After running this your APIhook Package should have custom procedure call.

or

Alternately you can use the following API to run the pre-processor

DECLARE

L\_API\_MODULE\_ID NUMBER := 1279;--=> API\_MODULE\_ID

BEGIN

HR\_API\_USER\_HOOKS\_UTILITY.CREATE\_HOOKS\_ONE\_MODULE (L\_API\_MODULE\_ID);

DBMS\_OUTPUT.PUT\_LINE('SUCCESS');

EXCEPTION WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('EXCEPTION : '||SQLERRM);

END;

/\*

HR\_API\_USER\_HOOKS\_UTILITY.CREATE\_HOOKS\_ONE\_MODULE (L\_API\_MODULE\_ID);

\*/

At this level, your user hook is implemented and should work as per your requirement.  
If you change your custom package body, you don’t need to run the pre-processor again but if you change the procedure signature, you will need to run the pre-processor again,

### **Delete custom code from APIuser Hook**

Get Hook Call ID and Object Version\_number

SELECT api\_hook\_call\_id,object\_version\_number

FROM HR\_API\_HOOK\_CALLS

WHERE call\_package = 'TGC\_USERHOOK\_PKG'

AND call\_procedure = UPPER('TGC\_ABS\_LEAVE\_VALIDATIONS')

SELECT \*

FROM HR\_API\_HOOK\_CALLS

WHERE api\_hook\_id = 3840

AND API\_HOOK\_CALL\_ID=1281;

BEGIN

Hr\_Api\_Hook\_Call\_Api.delete\_api\_hook\_call ( FALSE, API\_HOOK\_CALL\_ID =>1281,object\_version=>3);

DBMS\_OUTPUT.PUT\_LINE('deleted Successfully');

END;

/\*

Hr\_Api\_Hook\_Call\_Api.delete\_api\_hook\_call ( p\_validate => FALSE,

p\_api\_hook\_call\_id => 12345,  
p\_object\_version\_number =>2  
);

\*/

IF you have Error

You have to complie These packages

ALTER PACKAGE HR\_PERSON\_ABSENCE\_BK1 COMPILE BODY ;

ALTER PACKAGE HR\_PERSON\_ABSENCE\_BK1 COMPILE PACKAGE ;

ALTER PACKAGE FND\_DATA\_SECURITY COMPILE PACKAGE ;

ALTER PACKAGE FND\_DATA\_SECURITY COMPILE BODY ;