Message handling in SAP CRM Web UI

Applies to:
SAP CRM 7.0. For more information, visit Customer Relationship Management homepage

Summary
This article is aimed at understanding different ways of handling messages while working with SAP CRM Web Client UI Framework.

Author: Rohit Khetarpal
Company: Deloitte Consulting India Pvt. Ltd.
Created on: 18 August 2011

Author Bio
Rohit Khetarpal is a consultant at Deloitte Consulting. He has more than 5 years of experience in working with SAP CRM and has worked in 5.0/6.0/7.0 (EHP1) releases of SAP CRM.
# Table of Contents

Introduction ........................................................................................................................................... 3  
Message types ....................................................................................................................................... 3  
Function Module for message processing ........................................................................................... 3  
   Adding Message ................................................................................................................................ 3  
   List of message related function modules ....................................................................................... 4  
Messages for Global Business Object Container ............................................................................. 4  
   Adding / Deleting message to Global Business Object Container .................................................. 4  
   Ways of adding to global message container .................................................................................. 4  
      Adding message at GENIL Level .................................................................................................... 4  
      Adding message at View Level .................................................................................................... 5  
Messages for Specific Business Object Container ............................................................................. 5  
   Adding message to specific business object container ................................................................... 5  
   Ways of adding to specific business object container .................................................................... 5  
      Setting message container for a specific object ......................................................................... 5  
      Adding message at GENIL Level .................................................................................................. 7  
      Adding message at View Level .................................................................................................... 8  
Messages using message service class ............................................................................................... 8  
Subscribing to messages ......................................................................................................................... 9  
Message Replacement .......................................................................................................................... 10  
   Configuration for message replacement ......................................................................................... 10  
   Working of message replacement concept ...................................................................................... 11  
Message filter and user parameters ..................................................................................................... 12  
   Adding message level to message .................................................................................................... 12  
   Adding user level to message .......................................................................................................... 12  
   Filter based on message level .......................................................................................................... 12  
   Filtering based on user level ........................................................................................................... 14  
   Filtering based on type of message .................................................................................................. 15  
Message Area on Web UI ...................................................................................................................... 15  
Related Content ................................................................................................................................... 16  
Disclaimer and Liability Notice ........................................................................................................... 17
Introduction

Messages play a key role when working with SAP CRM Web UI to provide information to the user about the behavior of application and actions to be performed if any. As a consultant, we commonly come across requirements where we are required to add / delete / read / replace messages on the Web UI or to use them to navigate to a view; messages are also used to add any business logic to be performed by subscribing to a message. SAP CRM provides different ways of working with messages based on the requirement such as scope of the message, severity, user who should be allowed to see the message and attaching messages to object types. In this article, we would see different ways of processing messages and how does framework handle them.

Message types

Messages that get displayed on Web UI can be broadly classified as below:

- Error
- Warning
- Information
- Success
- Abort
- Exit

Constants for each of these types are defined in interface IF_GENIL_MESSAGE_CONTAINER

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Level</th>
<th>R</th>
<th>Typing</th>
<th>Associated Type</th>
<th>Description</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT_ERROR</td>
<td>Const.</td>
<td>Type</td>
<td>SYMGTY</td>
<td></td>
<td>Message Type</td>
<td>'E'</td>
</tr>
<tr>
<td>NT_WARNING</td>
<td>Const.</td>
<td>Type</td>
<td>SYMGTY</td>
<td></td>
<td>Message Type</td>
<td>'W'</td>
</tr>
<tr>
<td>NT_INFO</td>
<td>Const.</td>
<td>Type</td>
<td>SYMGTY</td>
<td></td>
<td>Message Type</td>
<td>'I'</td>
</tr>
<tr>
<td>NT_SUCCESS</td>
<td>Const.</td>
<td>Type</td>
<td>SYMGTY</td>
<td></td>
<td>Message Type</td>
<td>'S'</td>
</tr>
<tr>
<td>NT_ABORT</td>
<td>Const.</td>
<td>Type</td>
<td>SYMGTY</td>
<td></td>
<td>Message Type</td>
<td>'C'</td>
</tr>
<tr>
<td>NT_EXIT</td>
<td>Const.</td>
<td>Type</td>
<td>SYMGTY</td>
<td></td>
<td>Message Type</td>
<td>'X'</td>
</tr>
</tbody>
</table>

Constants for message types are also defined in Include program CRM_MESSAGES_CON.

Function Module for message processing

Adding Message

CRM_MESSAGES_COLLECT function module is used to add messages to the log. This way is mostly used in scenarios where we have event based data processing such as through action profiles; also, it can be used to register messages from view level. Below is a sample code for the same. Object names as passed in iv_caller_name parameter are present in table CRMC_OBJECTS, use of any other object name may result in short dump.

```plaintext
CALL FUNCTION 'CRM_MESSAGE_COLLECT'
EXPORTING
  iv_caller_name = 'ORDERADM_H'
  iv_ref_object  = iv_ref_guid
  iv_ref_kind    = 'A'
  iv_msgno      = '000'
  iv_msgid      = 'ZTEST'
  iv_msgty      = 'I'
```
List of message related function modules

Following are the commonly used function modules to process messages:

<table>
<thead>
<tr>
<th>FM Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRM_MESSAGES_REGISTER</td>
<td>Gives updated application log which could be used in FM CRM_MESSAGES_COLLECT</td>
</tr>
<tr>
<td>CRM_MESSAGE_COLLECT</td>
<td>Message collection of various objects - object names are present in table CRMC_OBJECTS</td>
</tr>
<tr>
<td>CRM_ORDER_READ</td>
<td>use with requested object name as 'MESSAGES' to get messages from object layer and fill buffer</td>
</tr>
<tr>
<td>CRM_MESSAGE_DISPLAY</td>
<td>to get messages from application log, provides message handle - which can be passed to FM CRM_MESSAGES_GET_INFO to get details of message</td>
</tr>
<tr>
<td>CRM_MESSAGES_GET_MSG_INFO</td>
<td>Provides detail of message</td>
</tr>
<tr>
<td>CRM_MESSAGES_DELETE</td>
<td>Used to disable or delete message</td>
</tr>
<tr>
<td>CRM_MESSAGES_SAVE</td>
<td>Save error log on database - internally calls FM BAL_DB_SAVE</td>
</tr>
</tbody>
</table>

Messages for Global Business Object Container

Adding / Deleting message to Global Business Object Container

Adding messages to global message container is one of the ways to display messages on UI. It can be used when there is a requirement for the message to be available globally as there is one instance of it that is active in the session and can be read using BOL Core message container manager. It can also be used to control the display of message only once using export parameter to its method and to reset all the messages at once. Also, it is useful in situation where we do not want to attach the message to a specific business object (BOL entity).

Ways of adding to global message container

Messages can be added to the container in presentation layer as well as BOL/GENIL layer as explained below.

Adding message at GENIL Level

In methods of genil class where we have iv_root_list is available as import parameter we can get global message container as below and add message using it as shown below:

```
Data: gr_gmsg type ref to CL_CRM_GENIL_GLOBAL_MESS_CONT.
gr_gmsg = iv_root_list->get_global_message_container( ).
gr_gmsg->add_bapi_messages(
   it_bapi_messages = it_msg
   iv_show_only_once = iv_show_only_once ).
```
To add messages in other methods we need to get BOL Core instance and get container instance and then add message to it. Below is standard code as used in GET_QUERY_RESULT method of CL_CRM_BOL_DQUERY_SERVICE. Flag iv_show_only_once is used to control displaying of message only once on Web UI.

```abap
lr_core = cl_crm_bol_core=>get_instance( ).
lr_message_cont = lr_core=>get_global_message_cont( ).
lr_message_cont=>add_message(  
    iv_msg_type = if_genil_message_container=>mt_error  
    iv_msg_id = 'CRM_BOL'  
    iv_msg_number = '017'  
    iv_show_only_once = abap_true  
    iv_msg_level = '3' ).
```

We can use implementation of method IF_GENIL_MESSAGE_CONTAINER~DELETE_MESSAGES to delete messages.

Adding message at View Level

Code as explained above can be used at view level as well. We can also utilize the attribute CORE as present in CL_CRM_BOL_ENTITY and then proceed as explained above.

### Messages for Specific Business Object Container

**Adding message to specific business object container**

To attach messages to specific BOL object, we can make use of specific business object container to add messages. To do this, we would have to get message container for the container object and add or delete message using that.

**Ways of adding to specific business object container**

**Setting message container for a specific object**

Setting of message container objects is generally done in genil class when the object is created or when the object is accessed as the requirement may be and this message object can then be used at other places where required.

Below is a sample code that shows how to first create a container and then set it for the object. This sample code is as used in IF_GENIL_APPL_INTLAY~GET_OBJECTS method of CL_CRM_BUIL class.
DATA: lv_message_cont   TYPE REF TO cl_crm_genil_bapi_mess_cont.
  lv_message_cont ?= lv_object->get_message_container( ).
  IF lv_message_cont IS NOT BOUND.
    CREATE OBJECT lv_message_cont.
    lv_object->set_message_container( lv_message_cont ).
  ENDIF.

We could also use to set our custom message containers and use it wherever required at view level as shown below:

  lr_core = cl.crm.bol_core->get_instance( ).
  lv_cont_manager = lr_core->get_message_cont_manager( ).
  lr_msg_container ?= lv_cont_manager->get_message_cont( iv_object_name = lv_object_name
                   iv_object_id   = lv_object_id  ).

  IF lr_msg_container IS NOT BOUND.
    CREATE OBJECT lr_msg_container.
    CALL METHOD lv_cont_manager->set_message_cont
      EXPORTING
        iv_object_name  = lv_object_name
        iv_object_id    = lv_object_id
        iv_message_cont = lr_msg_container.
  ENDIF.

Once we set message container - we can use the same to get/create/delete messages based on the requirement.

It is also important to note that in standard we set message containers for root/access objects and while reading containers from entity also it checks if it is access object, otherwise it tries to get its parent to reach there and get its message container. Below code shows how it reads entity message container in method GET_MESSAGE_CONTAINER of class CL_CRM_BOL_ENTITY:

  LV_CONT_MANAGER = ME->CORE->GET_MESSAGE_CONT_MANAGER( ).

  * find next access entity where entity belongs to
  LV_ACCESS_ENT = ME.
  try.
    while LV_ACCESS_ENT is bound and
    ME->OBJECT_MODEL->IS_ACCESS_OBJECT( LV_ACCESS_ENT->MY_MANAGER_ENTRY->OBJECT_NAME ) = ABAP_FALSE.
      LV_ACCESS_ENT = LV_ACCESS_ENT->GET_PARENT( ).
  end.}
endwhile.

    check LV_ACCESS_ENT is bound.
    RV_RESULT =
    LV_CONT_MANAGER->GET_MESSAGE_CONT(
    IV_OBJECT_NAME = LV_ACCESS_ENT->MY_MANAGER_ENTRY->OBJECT_NAME
    IV_OBJECT_ID   = LV_ACCESS_ENT->MY_MANAGER_ENTRY->OBJECT_ID).
    catch CX_CRM_UNSUPPORTED_OBJECT.                    "#EC NO_HANDLER
    * cannot happen
endtry.

Adding message at GENIL Level

In standard code, this is generally used in Read/Modify methods of Genil Class where we get container
object and find its message container. Below is example of standard code for BuildHeader object.
It is written in Modify method of CL_CRM_BUILHEADER class.

```
DATA: lv_objlist      TYPE REF TO if_genil_cont_root_objectlist,
      lv_partner      TYPE REF TO if_genil_cont_root_object,
      lv_msg_cont     TYPE REF TO cl_crm_genil_bapi_mess_cont.
* casting - to identify the interface we are working with.
TRY.
*     we expect a list of root objects
  lv_objlist  = iv_ref.
  CATCH: cx_sy_assign_cast_error.
*     else exit: should never happen
  EXIT.
ENDTRY.
  lv_partner  = lv_objlist->get_first( ).
  lv_object_id  = lv_partner ->get_object_id( ).
  lv_msg_cont   = lv_partner->get_message_container( ).
    lv_msg_cont->add_messages( iv_object_name   = me->object_name
                                iv_object_id     = lv_object_id
                                it_bapi_messages = lt_ci_return ).
```

We can use implementation of method IF_GENIL_MESSAGE_CONTAINER~DELETE_MESSAGES to
delete messages.
Adding message at View Level

To add a message to a specific object container, we need to get an instance of that container using BOL core. We can get an instance of the message container from the entity by calling the `GET_MESSAGE_CONTAINER` method of the entity class `CL_CRM_BOL_ENTITY` and then add the method as explained in the previous step. Below is a code snippet for the same.

```java
DATA: lr_msg_container TYPE REF TO cl_crm_bol_entity.
    CALL METHOD lr_msg_container->add_messages
    EXPORTING
        iv_object_name = lv_object_name
        iv_object_id = lv_object_id
        it_bapi_messages = li_message.
```

We can utilize different methods of the interface `IF_GENIL_MESSAGE_CONTAINER` based on requirements. Also, there is an abstract class implementation - `CL_CRM_GENIL_ABSTR_MESS_CONT` of this interface which further has subclasses to process messages as required and can be utilized as per the requirement.

We can use the implementation of method `IF_GENIL_MESSAGE_CONTAINER~DELETE_MESSAGES` to delete messages.

Messages using message service class

At presentation layer level, we can access an instance of message service class from `view_manager` attribute and call the `add_message` method to add a message. Below is an example:

```java
DATA: lr_msg_service TYPE REF TO cl_bsp_wd_message_service,
    lr_msg_service ?= view_manager->get_message_service( ).
    lr_msg_service->add_message( iv_msg_type = 'I'
        iv_msg_id = 'BSP_WD'
        iv_msg_number = '007' ).
```

Also, at places where we do not have that instance available, we can call `get_instance( )` method of message service class and then use the `add_message` method to add a message. Below is an example:

```java
DATA: lr_msg_service TYPE REF TO cl_bsp_wd_message_service,
    lr_msg_service ?= cl_bsp_wd_message_service->get_instance( ).
    lr_msg_service->add_message( iv_msg_type = 'I'
        iv_msg_id = 'BSP_WD'
        iv_msg_number = '007' ).
```
Subscribing to messages

We can subscribe to a message and handle when this message is added using method HANDLE_MESSAGE of interface IF_BSP_WD_MESSAGE_HANDLER, here we can write code to navigate to any navigation link added to the message or do any other processing as required. This is used in standard code as well to subscribe to a common message that appears on UI when you try to navigate without saving:

This message is subscribe at view manager level in the class CL_BSP_WD_VIEW_MANAGER in the method DO_FINISH_INPUT as in the code below:

It is also used in Interaction Center scenarios to navigate to specific views based on conditions such as account not confirmed.

It is also used to navigate from messages appearing in alert area in interaction center scenarios to any view. Below is the code as present in method IF_BSP_WD_MESSAGE_HANDLER~HANDLE_MESSAGE of class CL_IC_INTERACTION_MGR_INTERNAL that handles message of message class CRM_IC_SERVICES.

```java
data: context_area type ref to cl_crm_ic_contextareaview_impl.
class cl_crm_ic_services definition load.
context_area = cl_crm_ic_services=>contextarea_contr.

case is_message-number.
  * Overlap warning
  when '100'.
    context_area->navigate( 'AlertToInteractionRecordView' )."#EC NOTEXT
endcase.
```
Message Replacement

Configuration for message replacement

We can replace messages appearing on Web UI with our own custom message through configuration.

It can be done in SPRO -> Customer Relationship Management -> UI Framework -> UI Framework Definition -> Define Messages to Be Replaced

SAP provides a 'DEFAULT' profile; we can create our own profile and add messages to be replaced.

Adding message replacement to the profile is shown below:

This profile has to be assigned to function profile of the business role, which means message replacements can be controlled at business role level. To do that, configuration can be done in SPRO -> Customer Relationship Management -> UI Framework -> Business Roles -> Define Business Role, select a business role and click on Assign Function Profiles - give function profile ID as MSG_REPLACE and profile value as given as shown above (DEFAULT or custom profile value).
Working of message replacement concept

Framework reads and sets message replacements that have been configured as explained above through the application controller class `CL_BSP_WD_APPL_CONTROLLER` in `DO_INIT` method by calling `GET_MSG_REPLACEMENTS` (of application controller class) and `SET_REPLACEMENTS` (of message service class) methods as shown below.

Method `GET_MSG_REPLACEMENTS` as in the above code reads message replacements as set in the configuration through application class `CL_CRM_UI_FRAME` of component `CRM_UI_FRAME` as shown below:

Internally `GET_BY_NAME` method it creates an instance of class `CL_BSP_WD_MSG_REPLACEMENTS` which calls its constructor to read configuration tables for replacement as shown below:
Message filter and user parameters

Adding message level to message

We can add filter to messages using message service class as shown below:

```java
prepare message
  lv_msg_service = cl_bsp_wd_message_service->get_instance ( ).
  lv_msg_type = if_genmsg_message_container->get_warning ( ).
  lv_msg_id = 'CRM_IC_APPL'.
  lv_msg_number = '003'.
  lv_msg_v1 = if_etti->get_relative_name ( ).
  lv_msg_v2 = if_exception->get_text ( ).
  lv_msg_v3 = lv_exc_incl_name.
  lv_msg_v4 = lv_exc_arg1_line.
  lv_msg_level = '9'.
```

Adding user level to message

For adding user level to messages that are handled using APIs, we use function module as shown below:

```
call function 'CRM_MESSAGES_PUT_USERLEVEL'
  EXPORTING
    lv_level = 'go_msglevel_professional'.
```

Internal it sets a user parameter as shown below:

```
  * Move the level to global data...
  QV_Level = lv_level.
  * ... and into the memory
  set parameter id 'CRM_USER_LEVEL' field QV_level.
endfunction.
```

Filter based on message level

Interface for getting filter for messages is IF_BSP_WD_STATECONTEXT and logic can be implemented in its method GET_MESSAGE_FILTER. This interface is implemented at Window/View controller classes. In case of BT (Business Transaction Classes - such as Lead/Opportunity) - window IMPL class disables filter by default and at runtime and it enables the same if an entity is present as shown in screenshot below, this method needs to be enhanced to implement custom logic, if any:
In message service class CL_BSP_WD_MESSAGE_SERVICE - filtering of messages is done at runtime in method APPLY_USER_FILTER

```java
method APPLY_USER_FILTER.
data: LV_MSG_LEVEL type BSP_WD_MESSAGE_LEVEL.
get parameter ID 'BSPWD_USER_LEVEL' field LV_MSG_LEVEL.
if sy-subrc ne 0.
  * user parameter does not exist -> use default
  LV_MSG_LEVEL = 3.
endif.
if LV_MSG_LEVEL is initial.
  * user parameter not set -> use default
  LV_MSG_LEVEL = 3.
endif.
if LV_MSG_LEVEL = 0.
  * at level 0 only important info messages pass
  delete ME-MESSAGE_TAB where SORT_TYPE ne IMPORTANT_USER_INFO."$SC_CI_SORTSEQ
  delete ME-MESSAGE_TAB where MESSAGE_LEVEL > LV_MSG_LEVEL."$SC_CI_SORTSEQ
endif.
```

Values are validated against domain BSPWD_MSGLEVEL as below:

<table>
<thead>
<tr>
<th>Domain</th>
<th>BSPWD_MSGLEVEL</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Description</td>
<td>Degree of Detail for a Message</td>
<td></td>
</tr>
</tbody>
</table>

We can set this parameter in transaction SU3 for individual users.

If we set this parameter BSPWD_USER_LEVEL to 9 - it will display messages with the technical information such as the message class, number and variables involved if you point cursor at the message.
Filtering based on user level

Filtering based on user level is done in the function modules to read messages from application log. Usage is shown below in FM CRM_MESSAGES_DISPLAY by passing it to the work area ls_mfil.

This work area is used in another FM call within CRM_MESSAGES_DISPLAY to search messages from log by passing user level to the function module BAL_GLB_SEARCH_MSG as shown below:

Values are stored in domain CRM_MSGLEVEL as below:

We can set this parameter in SU3 depending upon what level of messages we want to display.
Filtering based on type of message

Filtering based on type of message (BSP Framework level/Application level/Specific BO level/Global BO level) is done in COLLECT_MESSAGES method of message service class CL_BSP_WD_MESSAGE_SERVICE as shown in code below.

```java
ME->FILTER = ME->VIEW_MANAGER->IF_BSP_WD_STATE_CONTEXT~GET_MESSAGE_FILTER(
  ME->FILTER ).
  if ME->FILTER~BSP_FRW_MESSAGES = ABAP_TRUE.
  if ME->FILTER~APPL_MESSAGES = ABAP_TRUE.
  if ME->FILTER~SPEC_BO_MESSAGES = ABAP_TRUE.
  if ME->FILTER~GEN_BO_MESSAGES = ABAP_TRUE.
```

Message Area on Web UI

Message area on Web UI gets created in the view ErrorView2.htm of the component BSPWD_BASICS by calling method CL_CHTMLCONFIGUTILITY~CREATE_MESSAGE_AREA.

This view reads all the messages by calling GET_MESSAGES method of its controller class which calls COLLECT_MESSAGES method of message service class CL_BSP_WD_MESSAGE_SERVICE. This method collects BSP messages; messages coming from applications/object layer, messages added to global or specific BO message containers and apply filters and message replacements to them/
Related Content

Message bar

SAP Customer Relationship Management
Best Practices for SAP CRM Web UI Customization
CRM Web Client UI Framework

For more information, visit Customer Relationship Management homepage
Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.