# Message handling in SAP CRM Web UI



# **Applies to:**

SAP CRM 7.0. For more information, visit <u>Customer Relationship Management homepage</u>

# **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

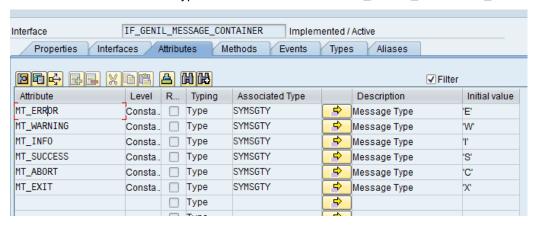
Messages play a key role when working with SAP CRM Web UI to provide information to the user about the behavior of application and actions to be performed if any. As a consultant, we commonly come across requirements where we are required to add / delete / read / replace messages on the Web UI or to 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



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.

```
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'
```

```
= 'message using CRM_MESSAGE_COLLECT'
  iv_msgv1
EXCEPTIONS
  not found
                          = 1
  appl_log_error
                          = 2
  OTHERS
                          = 3.
```

### List of message related function modules

Following are the commonly used function modules to process messages:

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

# **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:

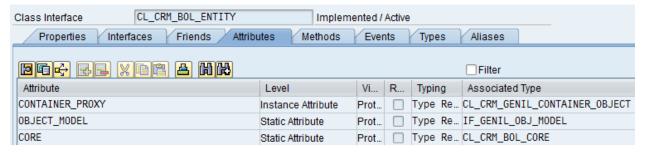
```
Data: gr gmsg type ref to CL CRM GENIL GLOBAL MESS CONT.
gr gmsg = iv root list->get global message container().
gr gmsg->add bapi messages(
    it bapi messages = it msg
    iv show only once = iv show only once ).
```

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.

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: Iv_message_cont TYPE REF TO cl_crm_genil_bapi_mess_cont.

Iv_message_cont ?= Iv_object->get_message_container().

IF Iv_message_cont IS NOT BOUND.

CREATE OBJECT Iv_message_cont.

Iv_object->set_message_container(Iv_message_cont).

ENDIF.
```

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

```
Ir_core = cl_crm_bol_core=>get_instance().

Iv_cont_manager = Ir_core->get_message_cont_manager().

Ir_msg_container ?= Iv_cont_manager->get_message_cont(iv_object_name = Iv_object_name iv_object_id = Iv_object_id).

IF Ir_msg_container IS NOT BOUND.

CREATE OBJECT Ir_msg_container.

CALL METHOD Iv_cont_manager->set_message_cont

EXPORTING

iv_object_name = Iv_object_name

iv_object_id = Iv_object_id

iv_message_cont = Ir_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().
```

```
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 BuilHeader 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 message to specific object container we need to get instance of that container using bol core. We can get instance of the message container from the entity by calling method GET\_MESSAGE\_CONTAINER of the entity class CL\_CRM\_BOL\_ENTITY and then add method as explained in above step. Below is code snippet for the same.

```
Ir_msg_container ?= Ir_entity->get_message_container().

CALL METHOD Ir_msg_container->add_messages

EXPORTING

iv_object_name = Iv_object_name

iv_object_id = Iv_object_id

it_bapi_messages = Ii_message.
```

We can utilize different methods of the interface IF\_GENIL\_MESSAGE\_CONTAINER based on requirements. Also, there is an abstract class implementations - 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 implementation of method IF\_GENIL\_MESSAGE\_CONTAINER~DELETE\_MESSAGES to delete messages.

# Messages using message service class

At presentation layer level, we can access instance of message service class from view\_manager attribute and call method add\_message to add a message. Below is an example:

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

# 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:

Details Required entry field not filled; click here to continue anyway

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:

```
DO FINISH INPUT
Method
                                                                          Active
    28
              register message handler for mandatory field check override
    29
                  ME->MESSAGE SERVICE->SUBSCRIBE MESSAGE( IV MESSAGE CLASS = 'BSP WD'
    30
    31
                                                           IV MESSAGE NUMBER = '007'
                                                           IV SUBSCRIBER = ME ).
    32
    33
                catch CX BSP WD DUPL MESS SUBSCR.
    34
              endtry.
```

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.

```
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.
```

SAP COMMUNITY NETWORK SDN - sdn.sap.com | BPX - bpx.sap.com | BOC - boc.sap.com | UAC - uac.sap.com

# Message Replacement

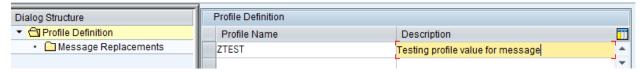
# Configuration for message replacement

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



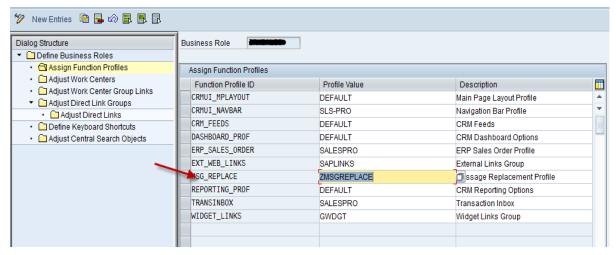
SAP provides a 'DEFAULT' profile; we can create our own profile and add messages to be replaced.



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.

```
Class Builder: Class CL_BSP_WD_APPL_CONTROLLER Display
🗧 🐤 | 🌱 😘 啫 🔞 | 🚰 🌴 🖳 🖒 | 몲 昼 🗀 且 | 🚱 🥵 Pattern Pretty Printer | Signature | 📲 🗐 Public Section 🛮 🗐 Protected Section 💆 Private Section
                  DO INIT
ethod
               if ME->WD_APPL is bound.
   71
                  get controller replacement table from application class
                 LOAD_REPLACEMENTS ( WD_APPL ).
                  get workarea definition
                 ME->IF BSP WD VIEW MANAGER~WORKAREA DEF = ME->WD APPL->GET WORKAREA DEFINITION().
   74
   75
                 get message replacements
                 data: LV_MSG_REPLACEMENTS type rel_to_CL_BSP_WD_MSG_REPLACEMENTS.
LV_MSG_REPLACEMENTS = ME->WD_APPL->GET_MSG_REPLACEMENTS().
   76
   77
                 ME->MESSAGE_SERVICE->SET_REPLACEMENTS( LV_MSG_REPLACEMENTS).
   78
   79
```

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:

```
Class Builder: Class CL_CRM_UI_FRAME Display
🗢 🔷 | 🦅 😘 喑 🎯 | 🚰 👭 🕮 🖒 | 品 🚊 💷 🚹 | 👼 🥦 Pattern - Pretty Printer - | Signature - | 📌 🗐 Public Section
                GET_MSG_REPLACEMENTS
/lethod
                                                                     Active
       method GET MSG REPLACEMENTS.
          data: LV_PROFILE type ref to IF_CRM_UI_PROFILE,
                LV_PROFILE_NAME type BSPWD_MSG_REPLACEMENT_PROFILE.
             LV PROFILE = CL_CRM_UI_PROFILE=>GET_INSTANCE().
              LV_PROFILE_NAME = LV_PROFILE->GET_COMPONENT PROFILE( 'MSG_REPLACE' ).
              RV RESULT = CL BSP WD MSG REPLACEMENTS=>GET BY NAME ( LV PROFILE NAME ) .
             if RV RESULT is initial.
              RV_RESULT = CL_BSP_WD_MSG_REPLACEMENTS=>GET_BY_NAME('DEFAULT').
   11
              endif.
           catch CX ROOT.
   12
   13
          endtry.
       endmethod.
   14
```

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:

```
Class Builder: Class CL BSP WD MSG REPLACEMENTS Display
🗢 🦈 | 🦅 😵 喑 🎯 | 🚰 🌴 🖷 🔥 | 옯 🖾 🖽 🏮 🖽 🔁 Rattern 💮 Pretty Printer | Signature | 🕸 🖼 Public Section
                CONSTRUCTOR
Method
       □ method CONSTRUCTOR.
         * check if profile exists
          select single PROFILE from BSPWD MSGREPL PF into ME->PROFILE where PROFILE = IV PROFILE.
           if IV_PROFILE = 'DEFAULT'.
              Note: profile DEFAULT is supported in any case
            else.
              raise NOT EXISTING.
            endif.
          endif.
    14
         * read SAP delivered replacements
          select REPLACED MSG ID REPLACED MSG NO REPLACING MSG ID REPLACING MSG NO
    16
    17
                 from BSPWD MSGREPL
    18
                  into corresponding fields of table ME->REPLACEMENTS
                 where PROFILE = IV PROFILE.
```

© 2011 SAP AG

# Message filter and user parameters

#### Adding message level to message

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

```
lr msg service = cl bsp wd message service=>get instance( ).
lv_msg_type = if_genil_message_container=>mt_warning.
lv_msg_id = 'CRM IC APPL'.
lv_msg number = '003'.
lv_msg_v1 = lr_rtti->get_relative_name().
lv_msg_v2 = ir_exception->get_text().
lv_msg_v3 = lv_exc_incl_name.
lv_msg_v4 = lv_exc_src_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
    iv level = gc msglevel-professional.
```

Internal it sets a user parameter as shown below:

```
CRM MESSAGES PUT USERLEVEL
Function module
   Attributes Import Export Changing Tables Exceptions Source code
     23
          * Move the level to global data...
     24
           gv level = iv level.
     25
           * ... and into the memory
            set parameter id 'CRM_USER_LEVEL' field gv_level.
     26
     27
     28
          endfunction.
```

# Filter based on message level

Interface for getting filter for messages is IF\_BSP\_WD\_STATE\_CONTEXT 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:

```
Class Builder: Class CL_CRM_BT_H_WINDOW Display
🗦 🖈 | 🤊 😘 ㎡ 🎯 | 🚰 🏋 🕮 砕 | 🚣 💆 💷 🚹 | 🍓 🧛 Pattern - Pretty Printer - | Signature - | 🕏
                IF_BSP_WD_STATE_CONTEXT~GET_MESSAGE_FILTER
      \label{lem:method_if_bsp_wd_state_context~get_message_filter.} \\
            lr_main_entity TYPE REF TO cl_crm_bol_entity.
         rv_filter = iv_filter.
         IF rv_filter IS NOT BOUND.
            CREATE OBJECT rv filter.
         ENDIF.
                 off special BO messages as default
                                                     -> This also clears entity list!
   12
         rv_filter->set_spec_bo_msg_active( abap_false ).
  14
          lr main entity = me->if bsp_wd_history_state_descr~get_main_entity( ).
   15
         IF 1r main entity IS BOUND.
   16
             leading entity is bound -> add to message filter
  17
18
            rv filter->set_entity_4_filter( lr_main_entity ).
      ENDMETHOD
```

In message service class CL\_BSP\_WD\_MESSAGE\_SERVICE - filtering of messages is done at runtime in method APPLY\_USER\_FILTER

```
Method
                 APPLY_USER_FILTER
                                                                       Active
       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 dos 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.
    13
    14
           endif.
    15
       if LV_MSG_LEVEL = 0.
    16
           at level 0 only important info messages pass
    17
             delete ME->MESSAGE_TAB where SORT_TYPE ne IMPORTANT_USER_INFO."#EC CI SORTSEQ
    18
    19
            delete ME->MESSAGE TAB where MESSAGE LEVEL > LV MSG LEVEL."#EC CI SORTSEO
    20
    21
```

Values are validated against domain BSPWD\_MSGLEVEL as below:



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.



© 2011 SAP AG

# 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.



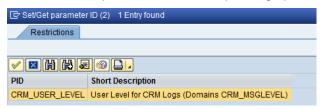
This work area is used in another FM call within CRM\_MESSAGES\_DISPLAY to search messages from log by passing user level to the function module BAL GLB SEARCH MSG as shown below:



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.



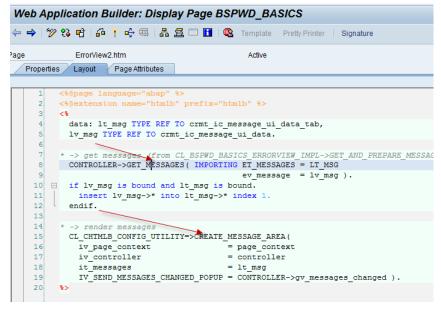
# Filtering based on type of message

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

```
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 CHTMLB CONFIG UTILITY=>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/

```
Class Builder: Class CL BSPWD BASICS ERRORVIEW BASE Display
🗢 🔿 | 🦅 😘 喑 🎯 | 🚰 🏋 🕮 🖒 | 🖧 🗸 🗀 🔟 | 🚱 🚳 Pattern | Pretty Printer | Signature | 🏕 🗏 Public Section 📑 Protec
lethod
                GET_AND_PREPARE_MESSAGES
                                                                     Active
      □ METHOD GET AND PREPARE MESSAGES .
          DATA: lv_msgsrv TYPE REF TO cl_bsp_wd_message_service.
          gv_messages_changed = abap_false.
         lv_msgsrv = me->view_manager->get_message_service().
         lv msgsrv->collect messages( iv message type
                                                            = if genil message container=>mt all
                             iv_delete_read_messages = iv_for_display ).
         DATA: lt_msg TYPE REF TO bsp_wd_message_tab.
          lt_msg = lv_msgsrv->get_messages( iv_msg_type
                                                                 = if genil message container=>mt all
                                          iv_delete_read_messages = iv_for_display ).
```

# **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 <u>Customer Relationship Management homepage</u>

# **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