# Utility ABAP classes

Several custom utility classes are available in S/4HANA. These utility classes are reusable and reduce the need to "reinvent the wheel" for basic, reoccurring needs in custom ABAP code. There are to be used to avoid unnecessary code duplication and slightly different implementation approaches.

All utility classes sit in package /SYQ/UTILITY and have the release contract C1 set to be able to be used in the ABAP Cloud standard.



## Amount in Words

This utility class simplifies the conversion of amounts into their written form in words including the currency, based on passed language key.

| **Class Name** | **/SYQ/CL_AMOUNT_IN_WORDS** |
|---|---|

| **Method Name** | **Description** | **Parameters** |
|---|---|---|
| GET_AMOUNT_IN_WORDS | Converts an amount into words in different languages | i_currency<br>i_amount<br>i_language<br>e_in_words |
| GET_AMOUNT_IN_WORDS_RU | Converts an amount into words in Russian | i_currency<br>i_amount<br>i_language<br>e_in_words |

## Application Logging

This utility class unifies the way we use the application log and has all needed functionality to create log entries.

| **Class Name** | **/SYQ/CL_APPLOG_UTILITY** |
|---|---|

| **Method Name** | **Description** | **Parameters** |
|---|---|---|
| GET_INSTANCE | Create an instance of the Application Logging Utility Class | |
| CREATE_LOG_OBJECT | Create a log object using header descriptors | i_object<br>i_subobject<br>i_external_id |
| ADD_FREE_TEXT | Add a free text value to Log | i_severity (Default: Default Severity)<br>i_text |
| ADD_MESSAGES_FROM_BAPI_RETURN | Add message table from BAPI Return | i_messages |
| ADD_MESSAGE_FROM_BAPI_RETURN | Add a single message in the format of BAPI Return | i_message |
| ADD_MESSAGE_FROM_SYST | Add message based on SY value (ie. SY-MSGID, SY-MSGNO, etc) | |

| | | |
|---|---|---|
| ADD_MESSAGE_FROM_BEHAVIOR | Add new message from RAP Behavior messages | i_behavior_message |
| SAVE | Save log object to DB | |
| GET_HANDLE | Get the handle ID of the created log object | r_handle |
| GET_LOG_OBJECT | Get the previous instance of the log object using handle | i_handle |
| READ_LOG_FROM_DB | Read the log results from DB using filter values | i_object<br>i_subobject<br>i_external_id<br>i_user<br>i_start_time<br>i_end_time<br>r_logs |

New message class /SYQ/APP_LOG added to maintain error messages while creation

New exception class /SYQ/CX_APPLOG_UTILITY created to handle errors in standard BAL processing

In terms of usage, the most common and straightforward processing would be: (a)

1. Get instance of the object
2. Create new log object
3. Add new text (executed one or more times depending on the number of messages)
4. Save

# Conversions

This utility class has the required logic to convert to and from strings, xstrings, and binaries.

| **Class Name** | **/SYQ/CL_CONVERSION_UTILITY** |
|---|---|

| **Method Name** | **Description** | **Parameters** |
|---|---|---|
| XSTRING_TO_STRING | Converts XSTRING using i_xstring_value to STRING | i_xstring_value<br>i_codepage (Optional defaulted to UTF-8)<br>i_replacement (Optional defaulted to "#")<br>r_string_value |
| STRING_TO_XSTRING | Converts STRING using i_string_value to XSTRING | i_string_value<br>i_codepage (Optional defaulted to UTF-8)<br>i_replacement (Optional defaulted to "#")<br>r_xstring_value |
| REPLACE_CHARACTER_IN_STRING | Replace all occurrences of a character (i_character) in string (c_string).<br>If i_ignore_case is set to false, it will only replace base on the replacement character case | i_character<br>i_replacement (Optional defaulted to space)<br>i_ignore_case (Optional defaulted to true)<br>c_string |

| | | |
|---|---|---|
| DECIMAL_TO_BINARY | Converts any decimal format of type integer to Binary string.<br><br>If i_output_signed is set to true, it would add a negative sign at the beginning of the string for any negative integer | i_decimal_value<br><br>i_output_signed (Optional defaulted to False)<br><br>r_binary_value |
| BINARY_TO_DECIMAL | Converts binary string to decimal | i_binary_string<br><br>r_decimal_value |
| XSTRING_TO_X255_TABLE | Converts xstring value to X255 (Raw with length 255) of table type | i_xstring<br><br>r_x255_Table |

# Sending of emails

This utility class unifies the email sending process in case it is part of a requirement.

| **Class Name** | **/SYQ/CL_EMAIL_UTILITY** |
|---|---|

| **Method Name** | **Description** | **Parameters** |
|---|---|---|
| CREATE_INSTANCE | Creates a new instance of the Email Utility | |
| SET_SUBJECT | Sets a free text subject | i_subject |
| SET_BODY_AS_HTML | Sets a free text body using HTML format | i_body_html |
| SET_BODY_BY_TEMPLATE_DATA | Sets the body through email template using a body data reference.<br><br>Optionally (if set as true), it will also set the subject based on the template's subject | i_template<br><br>i_data<br><br>i_set_subject_flag (Optional) |
| SET_BODY_BY_TEMPLATE_KEY | Sets the body through email template using key fields. The method will select data from template's CDS view and retrieve the data using the key.<br><br>Optionally (if set as true), it will also set the subject based on the template's subject | i_template<br><br>i_key_table<br><br>i_set_subject_flag (Optional) |
| SEND_EMAIL | Sends email | i_sender (Optional)<br><br>i_recipients<br><br>i_attachments (Optional)<br><br>i_signature_template (Optional) |
| FREE | Free up the instance | |

With regards to its usage, the straightforward usage is:

1. Create Instance via CREATE_INSTANCE
2. Set the Subject
3. Set the Body
4. Send Email via SEND_EMAIL
5. Free Instance via FREE

The subject can either be set based on the email template defined or by free text.

The body can also be set based on email template defined or by free text. When email template is used, use either of the two methods:

1. Set Body by Template Key - set the key field value and the internal method will select data directly from the CDS view based on the key fields
2. Set Body by Template Data - set a filled-up structure with data.

The default sender is currently configured within the SMTP Configuration and it will be picked up by the email utility. If there's no specific requirement to send via a sender, use the default sender instead.

Current controls setup in the SMTP configuration

1. Allowed Recipient Domain - in the test environment, the domain **syensqo.com** is whitelisted as an allowed domain. Any other recipient domain will not be accepted during testing phase.
2. Allowed Sender Domain - Domain **syway.syensqo.com** is whitelisted as the only allowed sender domain in the system as per Syensqo requirement. No other domains will be allowed to send email
3. Default Sender - it is currently setup as donotreply@syway.syensqo.com

The following controls are configured via Fiori application or IMG. The IMG path is IMG -> ABAP Platform -> Application Server -> System Administration -> Email Configuration

Default Signature is currently setup to /SYQ/DEFAULT_SIGNATURE Email template. This can be overridden using the parameter i_signature_template in the SEND_EMAIL method.

# Enterprise Structure Constants

This utility class provides a unified access to enterprise structure constants.

| **Class Name** | **/SYQ/CL_CONSTANT_UTILITY** |
|---|---|

| **Method Name** | **Description** | **Parameters** |
|---|---|---|
| GET_CONSTANT_VALUE | Get a specific constant | i_constant_name<br>i_constant_type |
| GET_VALUES_FROM_TYPE | Get a range of constants based on type or grouping id | i_constant_type<br>i_group_id |

The table containing the ES constants is /SYQ/CONSTANTS. It includes the following constants: Company Code, Sales Org, Purchase Org and Plants.

Table maintenance is via Custom Business Configuration.

**No ES values are to be hard coded and only to be read from the table using the utility class.**

# Dynamic Logo for form outputs

This utility class provides the logic to determine and retrieve the required logo to be output on a form and removes the need to maintain it in each form build.

| **Class Name** | **/SYQ/CL_DYNAMIC_LOGO_UTILITY** |
|---|---|

| **Method Name** | **Description** | **Parameters** |
|---|---|---|
| GET_COMPANY_LOGO_FROM_BUKRS | Retrieve the company logo path and binary | i_bukrs<br>e_logo_path |
| GET_FILE_FROM_MIME | Retrieve logo binary from MIME repository | i_mime_path<br>e_logo_binary |

Logos are maintained in the MIME Repository, transaction code SO2_MIME_REPOSITORY. Folder is under SAP->PUBLICSYENSQO. Default logo file is called Default_Logo.png. It is only needed to add additional and different logos than the default.

Table /SYQ/T_LOGO_PATH contains the logo location path per company code. Company code 0000 is set up for the default logo.

# Launchpad Notifications

This utility class provides a simplified and unified approach to trigger custom launchpad notifications. Functionality provided is the sending of notifications with or without navigation targets.

| Class Name | /SYQ/CL_NOTIF_PROVIDER |
|---|---|

| Method Name | Description | Parameters |
|---|---|---|
| CREATE_NOTIFICATION | Creates a launchpad notification | i_type_key<br>i_recipient<br>i_parameter<br>i_nav_parameters |

- Message class /SYQ/NOTIFICATION to be updated to add a new notification text with parameters.
- Table /SYQ/T_NOTIF_CNF to be updated with the Notification type and created message ids to build up the notification header and body.

  Unique type key is required for each notification type with title and content text ids assigned. Sample table entry is as follows.

| TYPE_KEY | TITLE | CONTENT |
|---|---|---|
| PPM_WF_LPD_NOTIFICATION | 001 | 001 |

- Table /SYQ/T_NOTIF_NAV to be updated with navigation target, if required, having the semantic object and action.
- In case of changes being applied to existing notification configurations, i.e. adding additional navigation, changing text or parameters, it is needed to clear the notification metadata cache via tcode /n. /IWNGW/H_CLEAR_NOTIF.

# Switch framework

This utility class provides the logic to switch on or off certain functionality in custom code in each system separately. Each enhancement requires the switch to be implemented to be able to quickly deactivate it in case of an issue. it is also useful to implement endless loops to allow background task debugging when active.

| Class Name | /SYQ/CL_SWITCH_FRAMEWORK |
|---|---|

| Method Name | Description | Parameters |
|---|---|---|
| CHECK_SWITCH | Returns a boolean based on a check in the switch table | i_enhancement_id<br>i_enhancement_step<br>r_check_value |

Table /SYQ/T_SWITCHES to be updated with the switch definition and activation indicator. Table to be maintained in each system/client where and when needed.

# Integration-triggered File Handler Framework

This utility addresses the Integration Suite initiated file handling processing in S/4HANA. In summary, there are specific requirements where Cloud Integration will extract files from third party systems and send the file to S/4HANA Application Server (or Network File Server). Then, Cloud Integration will publish a message to a SAP Event Mesh queue and S/4HANA, subscribed to the queue, will receive the notification from Cloud Integration. S4HANA, through the framework, will process the said file based on the interface specific requirement

## Step 1. Create an application specific class with Interface

| Interface Name | /SYQ/IF_FILE_ITG_RUN |
|---|---|

| Method Name | Description | Parameters |
|---|---|---|
| RUN | Executes application specific requirements | i_file<br>i_path |

### Step 2. Configure the application specific class in the File Integration Configuration using Business Configuration

| Fields | Description |
|---|---|
| Interface ID | Unique Interface identifier (I.e. ERP-762) |
| Class Name | Newly create application specific class (see step 1) |
| Description | Free text description |

## TM Utility Class

This is TM utility class is designed to reduces code duplication across different enhancements and development objects related to Transportation Management.

| Class Name | /SYQ/CL_TM_UTILITY |
|---|---|

| | | |
|---|---|---|
| **Method Name** | **Description** | **Parameters** |
| GET_TOR_ID | Get TOR ID based on Key | I_TOR_KEY<br>E_TOR_ID |
| GET_TOR_ALTERNATE_KEY | Get TOR KEY based on TOR ID | I_TOR_ID<br>E_TOR_KEY |
| GET_TOR_DOCUMENT | GET TOR ID based on base document SO/ STO | E_TOR_KEY |
| GET_BASE_DOCUMENT | GET Base document number based on TOR ID | I_BASE_DOC_ID<br>E_TOR_ROOT |
| GET_TOR_ALL_NODES | TOR All nodes | I_TOR_KEY<br>I_BEFORE_IMAGE<br>E_ROOT<br>E_ITEM<br>E_MAIN_ITEM<br>E_STOP<br>E_STOP_SUCC<br>E_DOCREF<br>E_EXEC<br>E_EXEC_TR<br>E_ASSIGNED_FU<br>E_FU_KEY_LINK<br>E_FSD_ROOT<br>E_FSD_KEY_LINK<br>E_PARTY |
| CHECK_CONDITION | Checking condition result as true or false | tba |
| FOR_EXECUTION_STATUS | Check the current execution status of Freight Order | E_EXEC_STAT |
| RAISE_DEPARTURE | Explicitly raise departure | E_MESSAGE<br>E_FAILED_KEY |
| GET_GEOCOORDINATE | Get Location Details | TBA |
| TRIGGER_CHECKIN | Trigger check in (at the plant pick up location ) | I_TOR_KEY<br>E_MESSAGE<br>E_FAILED_KEY |

With regards to its usage, the straightforward usage is:

1. Get SO/ STO numbers based on Freight order or Booking.
2. Raising Events specifically Departure
3. Printing BOL/ CMR
4. Distance and Duration calculations or routes
5. Geo- Coordinate based on location id
6. All TOR nodes details.