SAP PRESS

SAP for Utilities

SAP Audit Management

POWERED BY SAP HANA

SAP S/4 HANA

A Business and Technical Roadmap to Deploying SAP

WHATSAPP +255738656506

INTRODUCTION

Welcome to the fascinating world of SAP. This book helps you crack the tricks of mastering SAP HANA Customization

Audit Management

Implementation Guide for Audit Management.

Note: For direct access to this implementation guide, use the transaction PLMC_AUDIT.

Tables, in which Customizing settings are made, have delivery class C if a deviating delivery class is not specifically mentioned. However, most of the tables in the settings for audit management have delivery class G. For G tables, you should create your own entries using the standard entries as templates and then delete the entries supplied by SAP. However, other recommendations may be made for individual tables. The advantage of the procedure described above is that, when a release update is performed, the current, and possibly improved, standard entries will again appear in your test client and these are then easy to compare with the existing entries.

Question List

In this section, you process the settings for the question list.

Questions lists are mainly used in audit management, but are also useful for other applications. Therefore, the table may contain entries for several applications. Before you change or delete entries, you should check to which application these entries belong.

Structure of List of Questions

In this process step, you define the hierarchical structure of question lists and the hierarchy levels at which valuations are performed.

Question list profiles can be subdivided over several hierarchy levels. Questions are possible for all hierarchy levels, but valuations are only possible for the levels that you define.

Standard settings

The tables of the view cluster have delivery class G.
For your own entries, use numerical keys in the namespace 1* to 4*.

Valuation Specifications

In this process step, you define valuation profiles and the associated valuation levels.

You define for each valuation profile, whether the valuation is to be qualitative (coded) or quantitative.

- For each qualitative valuation profile, you create codes in a values set in a subordinate valuation table, and you determine the optimum and minimum points, as well as the number of points for the codes. On the basis of the number of points, you can assign a quantity to a qualitative valuation. You can flag a code as a proposed value for the valuation.
- For each quantitative valuation profile, you define the scale area, the scale increments, and a valuation proposal on a detail screen. You can flag a scale value as a proposed value for the valuation.

In the valuation profile, you can reverse the scale direction for the valuation, that is, you can define that the least number of points or the lower end of the scale represent the optimum result.

Standard settings

The tables of the view cluster have delivery class G.
For your own entries, use keys in the namespace 1* to 4*, and A* to Z*.

Recommendation

Do not make any changes to the audit profiles once these have been used, because this could lead to error interpretations for audits that have been completed. If necessary, replace these profiles with changed profiles.

Priorities

In this process step, you define priorities for objects in the question list. These priorities can, for example, be used during the valuation of these objects.

Standard settings

This table has delivery class G.
For your own entries, use keys in the namespace 1* to 4*, and A* to Z*.

Control

In this section, you process the settings for audit flow control.

Audit Components: Properties, Forms

In this step, you define the properties of the following predefined audit data object types:

- Audit plan
- Question list
- Audit
- Corrective action

You can determine the following properties for each of these data object types:

- Change documents are written
- An authorization group must be assigned to the data objects when they are created
- The data objects have an approval requirement
- Form for the printing of a data object
- Status profile with the user statuses that the data objects are to run through
- Document link via Knowledge Provider (KPro)

Standard settings

This table has delivery class G.
No specific namespace has been reserved for the user.

Recommendation

Do not make your own entries in this table. Instead, adapt the entries that were delivered with the standard system.

Activities

Adapt Forms

SAP delivers the following forms for the individual data object types:

- Question list: PLM_QUEST
- Audit plan: PLM_AUDITLAN
- Audit: PLM_AUDIT
  This form is also used for the audit report.
- Action: PLM_AUDITACTION
  For the overview of the actions for an audit: PLM_AUDITACTION_L

Further notes

The forms available in the standard system also describe the interfaces for data transfer.

In Notes 432642 and 432643, you can find the current instructions for the adaption of forms (432642) and for the settings necessary for the transfer of PDF documents via the Internet (432643).

Define Number Range Intervals for BAdI Implementation

Use

For the elements of an FMEA and an audit, you can define that sequential external numbers are generated automatically when an object is created. You can define different number range intervals, which you then assign to the individual object types.

Requirements

There must be an active implementation for the BAdI PLM_AUDIT_IDENTIFIER:

- BAdI implementation using your own implementation class
  To be able to use automatic number assignment, a call of the static method "GENERATE_NEW_EXTERNAL_ID" from the example implementation class "CL_EXM_IM_PLM_AUDIT_IDENTIFIER" must be implemented in the method "IF_EX_PLM_AUDIT_IDENTIFIER~GET_EXTERNAL_ID".
- BAdI implementation using example implementation class "CL_EXM_IM_PLM_AUDIT_IDENTIFIER"
  No other steps are required to use automatic number assignment.

Activities

1. Define the number range intervals.
2. In Customizing activity *Audit Components: Properties, Forms*, assign a number range interval to the different object types.

Authorization Groups

In this process step, you define authorization groups. These authorization groups can or must be entered in audit components.

Requirements

If you want to use authorization groups, you must assign authorization profiles that contain the authorization object AUDIT_AUTH to the users.

Partner Functions

In this process step, you determine the roles that are available in the system for the partners involved in the audit. You can display these roles using the F4 input help.

Standard settings

This table has delivery class E.
For your own entries, use keys from the namespace 1* to 4*, and A* to Z*.

Recommendation

You should use the roles available in the standard system to ensure that you will have access to the corresponding authorizations. If necessary, adapt the descriptions of the roles to suit your requirements.

Audit Components: Define Residence Time

Use

In this work step, you define the residence times of the following, mandatory audit components:

- Question list
- Audit
- Audit plan

Standard settings

The table has the delivery class G.
There is no dedicated namespace reserved for the user.

Activities

Do not create any entries in this table. Process the residence times for the entries available in the standard system.

Digital Signature

Settings in User Maintenance

Use

In this Customizing activity, you specify the data in the user master record of the person who is to execute the signature. You enter the full name of the user and their time zone. When a signature is executed, the

system copies the signatory name together with the local time according to the signatory's personal time zone to the signed document.

Note

You must enter the user data in the productive system, that is, in each system in which the digital signature is to be available, as the settings in the user master cannot be transported.

Caution

Each user can maintain his or her address data and default values under *System -> User Profile -> Own Data*. This data includes the general user settings and the SFF settings for the user. If you are working with the digital signature, you should therefore not allow all users authorization to maintain their own data.

Requirements

Basic Information for Digital Signature

Activities

Specify the general user settings:

1. To do this, either execute the Customizing activity or choose *Tools -> Administration -> User Maintenance* in the SAP Easy Access menu.
2. Enter the user ID of the user whose data you want to maintain and choose *Change*.
3. Choose the *Address* tab and enter the user's first and last name.
4. Choose the *Defaults* tab and enter the user's personal time zone.
5. Choose *Save*.
6. If you want to use the **user signature**, you must also enter the **SSF information** in the user master for the user who is to execute a digital signature:
   a) Choose the *Address* tab of the user.
   b) Choose the *Other Communication* pushbutton and double-click the SSF entry to open the dialog box for maintaining SSF addresses.
   c) Enter the SSF information for the security product used.
   How the entries must be structured depends on the security product you are using. For more information, see the document *Maintaining User SSF Information* in the SAP NetWeaver documentation.
   d) Choose *Copy* and save your entries.

Signature Strategy

Define Authorization Groups

Use

In this activity, you define the authorization group that can provide a specific individual signature when executing signature strategies.

You use user groups if individual signatures are to be executed in succession by different groups of users, such as production operators, shift managers, and QM employees, and this process is to be implemented with a signature strategy. This has the advantage that an individual signature is only executed when the system has verified that the correct user group was assigned to the user for this individual signature.

You use the authorization groups to restrict the authorization for executing digital signatures in applications as follows:

- You define different authorization groups for users with different areas of responsibility.
- In the user master record, you assign authorizations for the authorization group that corresponds to the user's area of responsibility (authorization object C_SIGN_BGR). Depending on the application, it may also be necessary to assign the authorizations for the C_SIGN authorization object.
- You define individual signatures that must be created by the user of a specific authorization group and use them in the signature strategies for the relevant application.

Activities

1. Determine which user groups or which areas of responsibility must be distinguished in your company.
2. Define an authorization group for each user group.

Example

In your company, specific input values must be entered by an employee and then verified by a second employee. Make the following settings:

- Define the authorization groups *EING (entry)* and *VERIF (verification)*.
- The employees that enter data receive authorization for the *EING* group and the employees that check data receive authorization for the *VERIF* group.
- Define two individual signatures for the relevant applications and assign the first to authorization group *EING* and the second to authorization group *VERIF*.

Further notes

Authorization groups are valid in all areas in which the digital signature with signature strategy is used. Therefore, before you change existing authorization groups or use them for your purposes, ensure that this will not lead to conflicts of interest with other areas.

Define Individual Signatures

Use

In this IMG activity, you define the digital individual signatures that must be executed by users in a specific authorization group.

You can then use the individual signatures as substeps of a signature strategy that is executed in the relevant application if you sign a process step or confirm an input value outside the permitted value range.

Requirements

You have defined authorization groups (see Defining Authorization Groups).

Activities

Define the individual signatures that must be executed in the relevant application when signing a process step or for an invalid input value.

Example

In your company, invalid input values must be signed by the employee responsible for the values and must be checked and signed by another employee. You define the following individual signatures:

- *S1* with authorization group *EING*
- *S2* with authorization group *VERIF*

Further notes

Individual signatures are also valid in other areas in which the digital signature is used. Therefore, before you change existing individual signatures or use them for your purposes, ensure that this will not lead to any conflicts of interest with other areas.

Define Signature Strategies

Use

In this Customizing activity, you define signature strategies. To do this, make the following settings:
- Group individual signatures of different user groups into one signature process. -Define which signature method is used when the signature process is executed.

In Customizing for the application you can then define which signature strategy is used to execute a process step.

Requirements

You have defined individual signatures (see Define Individual Signatures).

Activities

To create a new signature strategy, execute the following activities:

A. Create new entry for the signature strategy

1. Choose *New Entries*.
2. Create the key, name, and signature method for the signature strategy.
3. Define whether the following options are required, possible, or disallowed:
   - *Comment*

Here, you can specify if the user has the option of entering a text comment when signing the application. You have the following options: *Disallowed* (no comment field is provided), *Possible* (comment field can be called up), and *Required* (signature is only possible after a comment has been entered).

- *Remark*

  Here, you enable the user to select a remark from a list of predefined remarks and add it to the signature. The availability of remarks depends on the respective application. You have the following options: *Disallowed*, *Required*, and *Possible*.

- *Document*

  Here, you specify if the document to be signed should also be displayed to the signatory. You have the following options: *Disallowed*, *Possible*, and *Required*.

- *Verification*

  Here, you allow signatures that have already been executed to be verified. A check is run in the background to determine if this document is still identical to the original document and if the previously executed signatures have been stored correctly in the system.

Note:

These settings are overridden if you chose the setting *Disallowed* for these options in the transaction SIGNO (Register Signature Objects for Digital Signature) for the application.

B. Assign individual signatures

In this step, you assign the individual signatures that can or must be executed when executing the strategy, to the signature strategy.

You can assign the same individual signature more than once or you can assign several individual signatures with the same authorization group. However, when executing the signature strategy, each user in the authorization group can only execute one signature.

When saving, the system specifies a counter for each assigned individual signature. The counter identifies the assignment within the signature strategy but has no influence on the signature sequence when the strategy is executed.

1. Select the signature strategy to which you want to assign individual signatures and choose *Assign Individual Signatures*.
2. To assign new signatures, choose *New Entries*.
3. Enter the key for the required individual signatures.
4. Save the settings.

C. Define the signature sequence

In this step, you define the sequence in which the individual signatures in a signature strategy must be executed.

You determine the signature sequence by selecting a predecessor for each individual signature, in a sequence matrix.

Then you can also list the predecessors for each individual signature. However, you cannot edit the data in the individual display.

Example:

Individual signatures S1, S2, S3, and S4 are assigned to a signature strategy. The signature sequences are specified in the matrix, as follows:

		Predecessor			
Signatures					
Signatures to Be Executed		**S1**	**S2**	**S3**	**S4**
	S1				
	S2	X			
	S3	X			
S4		X	X		

This means:

- S1 must be executed first.
- S2 and S3 can be executed after S1.
- S4 can be executed as soon as S3 is available.

Defining signature sequences in the matrix:

1. In the *Define Signature Strategy* view, select the signature strategy for which you want to specify the signature sequence.
2. Choose the *Signature Sequence* pushbutton.
   The system displays the dialog box for the signature sequence of the signature strategy. The dialog box shows the matrix of the individual signatures.
3. In each row, flag the direct predecessors of the individual signature listed at the start of the row.
4. Choose *Enter*.
   If necessary, the system adds the indirect predecessors for the signatures.
5. Save the settings.

Displaying predecessors for individual signatures:

1. Select the signature strategy for which you wish to display data.
2. Choose *Assign Individual Signatures*.
3. Select the individual signatures for which you wish to display the predecessor.
4. In the navigation area, choose *Display Predecessor*.

D. Define release statuses

In this step, you define one or more alternative release statuses for a signature strategy. For each release status, you determine a combination of individual signatures with which the signature process can be completed.

A signature process is only complete if all the signatures belonging to a release status have been executed. If you do not define a release status for a signature strategy, all the individual signatures in a signature strategy must be executed in each signature process.

To define a release status, you flag the required signature combinations in the overview of possible signature combinations.
You can then also list the individual signatures for each release status. However, you cannot edit the data in the individual display.

Example:

The following signature sequence is defined in a signature strategy:

	Predecessor Signatures			
Signatures to Be Executed	**S1**	**S2**	**S3**	**S4**
S1				
S2				
X **S3**				
X				
S4	X	X		

In the tabular overview, select the following release statuses:

Individual Signatures Required for Release

S1	S2	S3	S4
X	X	X	
X		X	X

The signature strategy can be completed with either S2 and S3 or with S4.
Flagging the release status in the tabular overview:

1. Select the signature strategy for which you want to define release statuses.
2. Choose *Display Release Statuses*.
   A table is displayed with all the possible signature combinations contained in the signature strategy.
3. Select the signature combinations that you want to use as the release statuses for the signature strategy.
4. Choose *Enter* and save the settings.

Displaying individual signatures for each release status:

1. Select the signature strategy for which you wish to display data.
2. Choose *Display Release Statuses*.
3. Select the release status whose data you wish to display and choose *Display Individual Signatures*.
   A list appears with the counters for the individual signatures that are assigned to the release status.

Further notes

Signature strategies also apply in other areas where digital signatures are used. Therefore, before you change existing signature strategies or use them for your purposes, ensure that this will not lead to conflicts of interest with other areas.

Audit Definition

In this section, you process the settings for the audit definition.

Audit Usage

In this process step, you define audit usages.

You can use audit management for a variety of purposes, for example, to perform quality audits or environmental audits. In general terms, you can apply audit management to any usages involving the valuation and appraisal of material or intangible items on the basis of a question list.

The entries in this table only serve to differentiate between audit usages. They have no controlling functions.

Standard settings

Caution! This table is client-independent. This means that the settings you make are immediately activated in **all** system clients.

This table has delivery class E.
For your own entries, use the keys 0* to 4*.

Recommendation

Do not delete the keys that are delivered in the standard version from SAP.
Reason: It is possible that the evaluation programs delivered by SAP access these keys.

Audit Type

In this process step, you define the audit types.

You specify an audit usage (for example, quality audit) and an audit category (for example, system audit, process audit, product audit) to provide a closer definition of the audit type. Note: The audit usages are defined in a Customizing table. The audit categories are predefined in the system.
These specifications only serve to differentiate between audit types. They have no controlling effect.

The following fields assigned to the audit type have a controlling effect:

- Status Profile
  You can supplement the predefined statuses (system statuses) that an audit runs through with user statuses.
- Approval Requirement

- Input Help for audit object

You can define one audit object for each audit type.

You define the audit object related to the audit type by determining a number with a short text for it. In addition, you can assign fields (data elements or objects) from the data dictionary to the audit object.
You can determine value sets in a table for fields that are used to define the audit components.

Standard settings

The tables of the upper two levels of the view cluster have delivery class G. For your entries, use the keys 1* to 4*, and A* to Z*.

Recommendation

Do not delete the entries delivered by SAP.
Reason: It is possible that programs delivered by SAP need to access these keys.

Rating

In this process step, you define the ratings.

Standard settings

The tables of the view cluster have delivery class G.
For your own entries, use the keys 1* to 4*, and A* to Z*.

Audit Trigger

In this process step, you define audit triggers.

Standard settings This table has delivery class G.

For your own entries, use the keys 1* to 4*, and A* to Z*.

System Adjustment

Text Management

Use

In this step you make settings for text processing with SAPscript.

You can define text types for the type object CGPL_TEXT, which you can assign to the individual objects of an investigation (audit, FMEA,...) in the BAdI PLM_AUDIT_TEXT_ID.

Standard settings

Text types are already defined; you must not delete these.

These text types are assigned to the objects in the default implementation of the BAdI.

Activities

Enter new text types and assign these to the objects in your own BAdI implementation. Use the default implementation as a copy template.

Coding Mask

In this process step, you can define the nomenclature and the corresponding coding masks used for the audit.

Coding masks simplify the display of complex keys (numbers) for audit components. Identifying numbers can be made easier to read by adding separators, for example, hyphens or blank characters. In this process step, you use masks to define how the numbers of audit components are to be structured with separators. One mask consists of several sections that are defined individually. Each of these sections begins with a separator.

In the audit management transaction, you can enter your identifying numbers without separators. The system then inserts the separators based on the mask that you have defined.

Using coding masks, the system can derive the number of a new, subordinate audit component from the number of the superordinate element.

This function is only supported for question list items and question list items for the audit.

Example

Mask definition: (0 = numeric, X = alphanumeric)

Key	**1. Sec.**	**2. Sec.**	**3. Sec.**	**4. Section**
AUD	.000	/XX	.00	- XXXX

Entry and formatting in audit management:

Entry: AUD47AB99ZUVB **Formatting:** AUD7/AB9-ZUVB

Requirements

Activities

1. First, define a key and an explanatory short text for the mask.
2. Decide whether or not you want the numbers of audit components to be automatically derived from numbers of superior elements.

Further notes

Maintain the individual sections of the mask as follows:

a) Select *New Entries*.

b) Enter the key of the mask and a short text.

c) Select the position of the first section that you want to define.

d) Define a separator.

e) Define the length of the section.

f) Decide whether the section is to contain numeric or alphanumeric characters.

g) Repeat steps c) to f), until you have defined all the sections of the mask.

Field Selection

In this process step, you change the field selection.

You can change the properties of the modifiable fields within the constraints pre-defined by SAP. Such changes to modifiable fields can be made dependent on the influencing fields.

The field selection is valid for a specific program and a specific screen group. For each program used in audit management, there is a corresponding maintenance function for the field selection. You can find the name of the program responsible for a screen using the F1 field help or using the status display in the system.

Microsoft Project Interface

In this process step, you define the assignment of the fields for the transfer of audit plans and audits to Microsoft Project.

You can use the Microsoft Project interface to transfer just audit plans or audit plans including the directly assigned audits to Microsoft Project. You can continue to process the audit components in Microsoft Project.

Within Microsoft Project, you can then create (as tasks) and process additional audits.

Activities

You need to define for the individual fields whether the field contents are to be exported or imported or both.

1. Check the field assignments proposed by the system.
2. To change an assignment, choose a different field in the *Microsoft Project field* column.

Further notes

For each field, the field type is defined as a text, date, or number field. Ensure that the field types in *audit management* and in *Microsoft Project* correspond to one another.

Functional Enhancements

In this process step you receive the instructions for the use of program interfaces for functional enhancements, so-called Business Add Ins (BAdI).
You can find additional information regarding this technology in the documentation for transactions SE1 or SE19.

The following BAdIs exist in the system:

- BAdIs for updating another system (for example, R/3 Enterprise) or own table
- PLM_AUDIT_AUO_UPDATE
  Create, change, and delete audit
- PLM_AUDIT_AUP_UPDATE
  Create, change, and delete audit plan
- PLM_AUDIT_COR_UPDATE
  Create, change, and delete action
- PLM_AUDIT_QUN_UPDATE
  Create, change, and delete question list
- General BAdIs (for example, authorizations, status management)
- PLM_AUDIT_AUTH_CHECK
  Activities in audit processing
- PLM_AUDIT_STATUS
  Status management for audit components
- PLM_AUDIT_GOS
  Generic object services for audit components
- BAdIs for the manipulation of individual audit components (question list , plan, ...)
- PLM_AUDIT_APPEARANCE
  Determine format of audit components
- PLM_AUDIT_ATTRIBUTES
  Process attributes of audit components
- PLM_AUDIT_IDENTIFIER
  Identification of audit components (number assignment)
- BAdIs especially for audits (audit object, audit valuation)
- PLM_AUDIT_OBJECT
  Definition/formatting of audit object
- PLM_AUDIT_CALCULATE
  Valuation of an audit (key figure calculation)
- BAdIs within the audit application
- PLM_AUDIT_HELP_LINKS
  Maintenance of help links

Standard settings

SAP delivers default implementations for the following BAdIs:

- PLM_AUDIT_AUTH_CHECK
- PLM_AUDIT_GOS
- PLM_AUDIT_HELP_LINKS

Recommendation

If object references are transferred with the BAdI interface methods, you should not delete or initialize these.

Only access public components in read mode.

Further notes

We will develop additional BAdIs in the future. You will be able to find these new BAdIs in the transactions SE1 and SE19 by searching for PLM_AUDIT*.

Status Profile for User Status

Use

You can use this activity to adjust the general status management to suit your requirements.

Activities

Proceed as described in the documentation for the activity Define Status Profile for User Status

.

Then assign the status profiles to the audit components or, in the case of the audit component "audit", to the audit type.

The status profile maintained in the audit type takes precedence over the status profile maintained for the audit component "audit".

Business Add-Ins (BAdIs)

General BAdIs

Activities in Audit Processing

Description

This BAdI contains the method "SET_AUTHORITY_ACTIVITIES". It is used to choose the possible activities for a user in audit processing.

You can find technical details about this method in the interface documentation IF_EX_PLM_AUDIT_AUTH_CHECK.

Example

- If activity "310" is assigned to a user, this user can evaluate the audit.
- If the activity "3214" is not assigned to a user, this user cannot confirm the audit.

Standard settings

In the transaction for definition of the BAdI "PLM_AUDIT_AUTH_CHECK", you can find the default coding delivered by SAP and the proposed activities.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

##

Status Management for Audit Components

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to change various functions within status management in audit management.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods:

- Check Whether Execution of Transaction Is Allowed
- Reading of Data for the Display
- Reading of Permissible Changes
- Determines Whether Master Data Changes are Allowed for Comp.
- Determines Whether a Component Can Be Used

Generic Object Services for Audit Components

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

This BAdI enables you to use the various functions of generic object services for audit management.

The following objects are linked to the generic object services:

- Question list
- Question list item
- Audit plan
- Audit
- Audit question/reply list
- Audit corrective action

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.

When the application program is executed, the system carries out the code in the method you wrote.

Example

See also:

Methods

Select the Services of Generic Object Services
##
#
########

Scheduling of Background Jobs

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to transfer job information that is required for background processing.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.

Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:

8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods

- GET_JOBLIST

BAdIs for Updating Other Systems or Individual Tables

Create, Change, and Delete Audit

Use

This Business Add-In is used in the Audit Management component (CA-AUD).

You can use this BAdI to enter all changes to an audit (create, change, delete) and the associated audit objects. You can then use it to replicate the changes in another system.

You can process changes at the following points in time:

- AT_SAVE
  The user has called the function 'Save' in the dialog. At this point in time, you can implement another check of the data to be saved. If errors are found during this check, this information can be returned. The data is not saved, and the user is informed.
- BEFORE_UPDATE
  The call is performed immediately before the data is transferred to the update. No messages can be output at this point. The save cannot be stopped.
- IN_UPDATE
  The call takes place in the update. No messages can be output at this point.

Requirements

Standard settings This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

The following methods are available:

- Create Audit When Saving
- Create Audit Before Update Task Is Called
- Create Audit in Update Task
- Create Audit Question When Saving
- Create Audit Question Before Update Task Is Called
- Create Audit Question in Update Task
- Create Audit Object When Saving
- Create Audit Object Before Update Task Is Called
- Create Audit Object in Update Task
- Change Audit When Saving
- Change Audit Before Update Task Is Called
- Change Audit in Update Task
- Change Audit Question When Saving
- Change Audit Question Before Update Task Is Called

- Change Audit Question in Update Task
- Change Audit Object When Saving
- Change Audit Object Before Update Task Is Called
- Change Audit Object in Update Task
- Delete Audit When Saving
- Delete Audit Before Update Task Is Called
- Delete Audit in Update Task
- Delete Audit Question When Saving
- Delete Audit Question Before update Task Is Called
- Delete Audit Question in Update Task
- Delete Audit Object When Saving

  Note

  This method is currently not in use. If necessary, use the methods DELETE_OBJECT_BEFORE_UPDATE, DELETE_OBJECT_IN_UPDATE, or another method called at the AT_SAVE point in time.
- Delete Audit Object Before Update Task Is Called
- Delete Audit Object in Update Task

Create, Change, and Delete Audit Plan

Use

This Business Add-In is used in the Audit Management component (CA-AUD).

You can use this BAdI or the corresponding methods to record all changes to an audit plan (create, change, display). You can then use it to replicate the changes in another system.

You can process changes at the following points in time:

- AT_SAVE

  The user has called the function *Save* in the dialog. At this point in time, you can implement another check of the data to be saved. If errors are found during this check, this information can be returned. The data is not saved, and the user is informed.
- BEFORE_UPDATE

  The call is performed immediately before the data is updated by the system. No messages can be output at this point. The save cannot be stopped.
- IN_UPDATE

  The call takes place during the update. No messages can be output at this point.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

Description

See also:

Methods:

- Create plan when saving
- Create plan before update task is called
- Create plan in update task
- Change plan when saving
- Change plan before update task is called
- Change plan in update task
- Delete plan when saving
- Delete plan before update task is called
- Delete plan in update task

##########

Create, Change, and Delete Actions

Use

You can use this Business Add-In (BAdI) or the corresponding methods to enter all changes to a corrective/preventive action (create, change, delete). You can then use it to replicate the changes in another system.

You can process changes at the following points in time:

- AT_SAVE
  The user has called the function *Save* in the dialog. At this point in time, you can implement still implement a check of the data to be saved. If errors are found during this check, this information can be returned. The data is not saved and the user is informed.

- BEFORE_UPDATE
  The call is performed immediately before the data is updated. No messages can be output at this point. The save cannot be stopped.

- IN_UPDATE

The call takes place during the update process. No messages can be output at this point.

Standard settings

This BAdI is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:

8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

See also:

Methods

- Create Action When Saving
- Create Action Before Update Task Is Called
- Create Action in Update Task
- Change Action When Saving
- Change Action Before Update Task Is Called
- Change Action in Update Task
- Delete Action When Saving
- Delete Action Before Update Task Is Called
- Delete Action in Update Task

##

Create, Change, and Delete Question Lists

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI or the corresponding methods to record all changes made to a question list and the related items (create, change, delete). can then use it to replicate the changes in another system.

You can process changes at the following points in time:

- AT_SAVE
  The user has called the function *Save* in the dialog. At this point in time, you can implement another check of the data to be saved. If errors are found during this check, this information can be returned. The data is not saved, and the user is informed.
- BEFORE_UPDATE
  The call is performed immediately before the data is transferred to the update. No messages can be output at this point. The save cannot be stopped.
- IN_UPDATE
  The call takes place in the update. No messages can be output at this point.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods:

- Create Header When Saving
- Create Header Before Update Task Is Called
- Create Header in Update Task
- Create Item When Saving
- Create Item Before Update Task Is Called
- Create Item in Update Task
- Change Header When Saving
- Change Header Before Update Task Is Called
- Change Header in Update Task
- Change Item When Saving
- Change Item Before Update Task Is Called

- Change Item in Update Task
- Delete Header When Saving
- Delete Header Before Update Task Is Called
- Delete Header in Update Task
- Delete Item When Saving
- Delete Item Before Update Task Is Called
- Delete Item in Update Task

##
##

XML Generation

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

This BAdI can be used to change the way in which the audit confirmation is displayed in HTML format. It can also be used to control the generation of the XML file when importing and exporting audit components.

Standard settings

The Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.

7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:

8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

Description

See also:

Methods:

- Add Data to Audit XML
- Determine XSLT File Name for Survey Generation
- Change XML File During Export - Change XML File During Import

ALV Functions

Description

You can use this BAdI to change different functions within the ALV Grid Control.

The following enhancements are supported:

- ADOPT_LAYOUT
  You can use this method to adapt the layout of the ALV Grid Control to suit your requirements.
- ADOPT_TOOLBAR_EXCLUDING
  You can use this method to adapt the inactive functions of the ALV Grid Control to suit your requirements.
- CHANGE_FIELDCATALOG
  You can use this method to change one individual line of the field catalog. However, this method is only called if you requested the change using the method IS_FIELDCAT_CHANGE_REQUESTED.
- CHANGE_GRAPHICS
  You can use this method to change the formatting properties of the ALV graphic connection. For example, you can name the X-axis or specify a title.

- CHANGE_HYPERLINKS
  You can use this method to change the hyperlink function of the ALV Grid Control.
- CHANGE_OUTTAB
  You can use this method to change one individual line of the output table. However, this method is only called if you requested the change using the method IS_OUTTAb_CHANGE_REQUESTED.
- DETERMINE_SAVE_MODE
  You can use this method to determine the save mode of the ALV Layout.
- IS_FIELDCAT_CHANGE_REQUESTED
  You can use this method to request that changes be made to individual lines in the field catalog.
- IS_OUTTAB_CHANGE_REQUESTED
  You can use this method to request that changes be made to individual lines in the output table.
- ON_BUTTON_CLICK
  This method is called when a pushbutton that was added using the method ON_TOOLBAR is chosen.
- ON_CONTEXT_MENU_REQUEST
  This method is called when you request the context menu. Additional functions can be added in the context menu.
- ON_MENU_BUTTON
  This method is called when a menu button that was added using the method ON_TOOLBAR is chosen.
- ON_TOOLBAR
  This method is called when formatting the toolbar. Additional functions can be added.
- ON_USER_COMMAND
  You can use this method to handle a user-defined function code.

Note

You can find additional information about the ALV Grid Control in the corresponding documentation in the SAP List Viewer (component BC-SRV-ALV).

BAdIs for Managing Individual Audit Components

Definition of Structure Types

Definition of Structure Types

Use

In this activity, you create structure types. For these structure types, you can create implementations for the BAdI BADI_PLM_AUDIT_STRUCTURE.

You can specify the structure type when you define the investigation type (audit type, FMEA type, and so on).

BAdI: Structural View for Investigations (Audit, FMEA...)

Use

This Business Add-In (BAdI) is used in the *Audit Management* component and other derived components, such as Failure Mode and Effects Analysis (FMEA).

You can use this BAdI to display investigations (audits, FMEA, and so on) and linked objects hierarchically. To do this, you open the *Hierarchy Display* view in the cockpit.

Requirements

You have stored a structure type for the investigation type (audit type, FMEA type, and so on).

Standard settings

The BAdI is filter-dependent and not designed for multiple use.
To fulfil your own requirements, define a new structure type and create an implementation for BADI_PLM_AUDIT_STRUCTURE.

In the standard system, the BAdI implementation PLM_AUDIT_STRUCTURE_00 is activated. This implementation allows you to display the history of the same investigation object for an investigation.

Activities

For information about implementing BAdIs as part of the Enhancement Concept, see SAP Library for SAP NetWeaver under BAdIs - Embedding in the Enhancement Framework.

See also

This BAdI uses the interface IF_EX_PLM_AUDIT_STRUCTURE.

For more information, display the interface in the Class Builder.

Definition of List Item Types

Define List Item Types

Use

In this activity, you create list item types. You can create implementations for the BAdI BADI_PLM_AUDIT_LIST_ITEM_TYPES for these list item types.

When you define the hierarchy profile, you can use the list item types defined here.

BAdI for Definition of List Item Types

Use

This Business Add-In (BAdI) is used in the *Audit Management* component and other derived components, such as the Failure Mode and Effects Analysis (FMEA).

You can use this BAdI to change the interface and the attributes for a list item type that is used in an investigation (audit, FMEA, and so on). You can assign list item types to the items of hierarchy profiles.

Standard settings

The BAdI is filter-dependent and not designed for multiple use.

In the standard system, the implementations of the list item types supplied by SAP are activated. There is no default code. In the implementation, you can change the screens and the functions for the list items or audit questions.

To use your own implementation, first create a list item type and then an implementation.

Activities

For information about implementing BAdIs as part of the Enhancement Concept, see SAP Library for SAP NetWeaver under BAdIs - Embedding in the Enhancement Framework.

See also

This BAdI uses the interface IF_EX_PLM_AUDIT_LIST_ITEM_TYPE.

For more information, display the interface in the Class Builder.

Definition of Action Types

Define Action Types

Use

In this activity, you create action types. You can create implementations for the BAdI BADI_PLM_AUDIT_ACTION_TYPE for these action types. With your own implementation of the BAdI you can create actions of the appropriate type.

BAdI for Definition of Action Types

Use

This Business Add-In (BAdI) is used in the *Audit Management* component and other derived components, such as Failure Mode and Effects Analysis (FMEA).

You can use this BAdI to influence the interface and behaviour of action types. You can control the use of action types using the BAdI BADI_PLM_AUDIT_ACTION_TYPE.

Standard settings

The BAdI is filter-dependent and not designed for multiple use.
In the standard system, the implementations of the action types supplied by SAP are activated. There is no default code. In the implementation, you can change the screens and functions of the action.

To use your own implementation, first create an action type and then an implementation for it.

Activities

For information about implementing BAdIs as part of the Enhancement Concept, see SAP Library for SAP NetWeaver under BAdIs - Embedding in the Enhancement Framework.

See also

This BAdI uses the interface IF_EX_PLM_AUDIT_ACTION_TYPE.

For more information, display the interface in the Class Builder.

Linkage of Objects with the Corrective Action

Use

This Business Add-In (BAdI) is used in the *Audit Management (CA-AUD)* component and in other derived components such as the Failure Mode and Effects Analysis (FMEA).

You can use this BAdI to create objects for a corrective action via the cockpit. The objects you create in this way are displayed under the corrective action in the tree structure.

For more information on the individual methods, see the documentation of the interface methods.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

The quality notification is being connected using the sample code.

To display the sample code, choose *Goto --> Sample Code --> Display*.

See also:

Methods:

Request Menu with Assignment Functions

Generate Objects

Request for Display Characteristics

Request for Context Menu Functions

Transfer Interface Function Code for Execution

##

Define Display of Audit Components

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

For the display of audit components, the data is converted from an internal to an external format. This conversion takes place before each output. The converted data can be, for example, displayed on the screen or printed. The formatting is initially performed by SAP, but you can change it to suit your requirements.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods

- Process Interface Structure of Question List Header
- Process Interface Structure of Question List Item
- Process Interface Structure of Audit Plan
- Process Interface Structure of Audit
- Process Interface Structure of Question List for Audit
- Process Interface Structure of Corrective/Preventive Actions

- Process Interface Structure of Audit Object
- Process Interface Structure of the Partner Assignment

##
#
##############################

BAdI for Functional Enhancement of Investigation Type (Audit Type,...)

Use

This Business Add-In (BAdI) is used in the *Audit Management* component and other derived components, such as Failure Mode and Effects Analysis (FMEA).

You can use this BAdI to define additional menu functions for an investigation (audit, FMEA, and so on) and its processing.

Standard settings

The BAdI is filter-dependent and not designed for multiple use.

You can create one implementation per investigation type.

In the standard system, BAdI implementations for some investigation types supplied by SAP are activated. There is no default code. Additional menu functions were included in the implementations.

Activities

For information about implementing BAdIs as part of the Enhancement Concept, see SAP Library for SAP NetWeaver under BAdIs - Embedding in the Enhancement Framework.

See also

This BAdI uses the interface IF_EX_PLM_AUDIT_TYPE.

For more information, display the interface in the Class Builder.

Edit Attributes of Audit Components

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to change attributes of audit components at different points in time.

You can:

1. Define additional criteria for the semantic correctness of the audit components.

2. Change the attributes when creating new components.

3. Perform additional activities when new components are created by copying existing components.

   1.)
   Whenever a change is made to an audit component, a check is performed to see whether or not the change is allowed. If you wish to extend this check, use this enhancement and transfer the results of your check in a return table as messages. These messages are later output in the log. If defect or cancellation messages were transferred, the relevant component cannot be saved. Current data and the data that has just been saved are available for the check.

   2.)
   When a new component is being created the attributes, for example, grouping, can be changed. Be careful when changing attributes.

   3.)
   When you are copying components, it is possible to make additional changes to the attributes. The relevant template is available for this.
   Note that when you are copying, you need to first call up the
   CHANGE_xxx_AT_CREATION method and then the CHANGE_xxx_ON_COPY method.
   Be careful when changing attributes.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

4. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.

5. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.

6. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.

7. Save your entries and assign the Add-In to a package.

8. To edit a method, double-click its name.

9. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.

10. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
    Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:

11. Choose ***Activate***.
    When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods

- Check of the Attributes of the Question List Header
- Check of the Attributes of the Question List Item
- Check of the Attributes of the Audit Plan
- Check of the Attributes of the Audit
- Check of the Attributes of the Question List for the Audit
- Check of the Attributes of the Corrective/Preventive Action
- Change Attributes of Question List Header When Creating
- Change Attributes of Question List Item When Creating
- Change Attributes of Audit Plan When Creating
- Change Attributes of Audit When Creating
- Change Attributes of Question List for Audit When Creating
- Change Attributes of Corrective/Preventive Action When Creating
- Change Attributes of Question List Header When Copying
- Change Attributes of Question List Item When Copying
- Change Attributes of Audit Plan When Copying
- Change Attributes of Audit When Copying
- Change Attributes of Question List for Audit When Copying
- Change Attributes of Corrective/Preventive Action When Copying
  ##

Identification of Audit Components (Number Assignment)

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to influence the following functions within Audit Management:

- Internal number assignment
- Description and identification of audit components within the navigation structure

- Change options for the key of an audit component

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods:

- Preassign Key When Creating Audit Component
- Define Display of Audit Component in Navigation Structure
- Can Key Be Changed ('0' -> Not Changeable)

Interfaces

IF_EX_PLM_AUDIT_IDENTIFIER.

##

Controlling Procedure for Audit Questions

Use

This Business Add-Inn (BAdI) is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to change audit questions automatically depending on other data (audits, audit questions). In particular, you can use it to set the relevance or execute the valuation of the audit question.

Standard settings

The implementations supplied by SAP are available for use.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

You can use the implementations supplied by SAP as sample code.

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods:

Returns Start Time of Procedure

Returns Whether Procedure Can Be Used

Change Audit Question

Assignment of Question Lists to Audit

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to influence the assignment of question lists to the audit, or the creation of an audit question using a template of a question list item.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods

- Changing the Attributes of the Audit Question List Item
- Check whether a question list can be assigned to the audit
- Check whether the question list can be assigned to the audit

Enhancements: Customer-Defined Fields

Customer-Defined Fields for Audit Plan

Use

This Business Add-In (BAdI) is used in the *Audit Management* component.

You can use this BAdI to add your own fields to the tab page for basic data or to your own tab page in the audit plan, without making any modifications.

Standard settings

The Business Add-In is not active in the standard system.

Activities

For information about implementing BAdIs as part of the Enhancement Concept, see SAP Library for SAP NetWeaver under BAdIs - Embedding in the Enhancement Framework.

See also

Interface

IF_EX_PLM_AUO_CUSTOM_FIELDS

Methods

Delivery of Dialog Data for Audit Plan

Set Dialog Data for Audit Plan

Conversion of Dialog Data into Database Format

Conversion of Database Data into External Dialog Structure

Check of Dialog Data for Audit Plan

Set Name for Customer-Defined Tab Page

Customer-Defined Fields for Audit

Use

This Business Add-In (BAdI) is used in the *Audit Management* component.

You can use this BAdI to add your own fields to the tab page for basic data, to the tab page for the result, or to your own tab page in the audit, without making any modifications.

Standard settings

The Business Add-In is not active in the standard system.

Activities

For information about implementing BAdIs as part of the Enhancement Concept, see SAP Library for SAP NetWeaver under BAdIs - Embedding in the Enhancement Framework.

See also

Interface

IF_EX_PLM_AUO_CUSTOM_FIELDS

Methods

- Delivery of Dialog Data for Audit
- Set Dialog Data for Audit
- Conversion of Dialog Data into Database Format
- Conversion of Database Data into External Dialog Structure
- Check of Dialog Data for Audit
- Set Name for Customer-Defined Tab Page

Customer-Defined Fields for Question List

Use

This Business Add-In (BAdI) is used in the *Audit Management* component.

You can use this BAdI to add your own fields to the tab page for basic data or to your own tab page in the question list, without making any modifications.

Standard settings

The Business Add-In is not active in the standard system.

Activities

For information about implementing BAdIs as part of the Enhancement Concept, see SAP Library for SAP NetWeaver under BAdIs - Embedding in the Enhancement Framework.

See also

Interface

IF_EX_PLM_QUN_CUSTOM_FIELDS

Methods

- Delivery of Dialog Data for Question List
- Set Dialog Data for Question List
- Conversion of Dialog Data into Database Format
- Conversion of Database Data into External Dialog Structure
- Check of Dialog Data for Question List
- Set Name for Customer-Defined Tab Page

Customer-Defined Fields for Corrective Action/Preventive Action

Use

This Business Add-In (BAdI) is used in the *Audit Management* component.

You can use this BAdI to add your own fields to the tab page for basic data or to your own tab page in the corrective/preventive action, without making any modifications.

Standard settings

The Business Add-In is not active in the standard system.

Activities

For information about implementing BAdIs as part of the Enhancement Concept, see SAP Library for SAP NetWeaver under BAdIs - Embedding in the Enhancement Framework.

See also

Interface

IF_EX_PLM_AUO_CUSTOM_FIELDS

Methods

- Delivery of Dialog Data for Corrective/Preventive Action
- Set Dialog Data for Corrective Action/Preventive Action
- Conversion of Dialog Data into Database Format
- Conversion of Database Data into External Dialog Structure
- Check of Dialog Data for Corrective Action/Preventive Action
- Set Name for Customer-Defined Tab Page

BAdIs for Audit Objects and Audit Valuations

Definition of an Object for Investigation (Audit,...)

Use

This Business Add-In Business Add-In(BAdI) is used in the *Audit Management, Failure Mode and Effects Analysis (FMEA)* and other components.

This BAdI contains the methods for checking, selecting and displaying the object for an investigation, depending on the investigation type. In Audit Management, we use the terms Audit Objects and Audit Types. The information below relates specifically to use in Audit Management but the description applies to investigations in general.

Since it is not possible to plan all possible audit objects in a standard software, and external data may have to be accessed, the audit object is connected using this interface. As a result, you can create your own implementations (in addition to those supplied by SAP) without having to make modifications.

Within this interface, you can access the detailed information about the audit object that was defined in Customizing for the audit type.

A filter type must be specified when this BAdI is implemented. This filter type is assigned in the field *Input Values for Audit Object (BAdI)* (technical field name PLMV_AUDIT_TYPE-FILTER) in Customizing for the audit type.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:

8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

Audit type: External product audit of product 4711 for the vendor IDES.

The audit object "Product 4711 for vendor IDES" is described by the two attributes **Product** (attribute type 01) and **Vendor** (attribute type 02). You define the associated data elements for the entry fields in Customizing. During the BAdI implementation, the system then accesses the data for the business partner and the product master record that exists in the system.

Audit type: Internal system audit in development AREA1 according to ISO 9000.

The audit object is described by the two attributes **QM System** and **Area**. The auditor therefore sees two entry fields for the audit object with those field labels that were defined in Customizing for the audit type. You define the input values for the possible QM systems in a separate value table in Customizing for the audit type. No value table exists for the individual areas in the system yet. During BAdI implementation, you access an external system.

To display sample code, choose *Goto -> Sample Code -> Display*.

Recommendation

The implementation must be in line with the Customizing entries for the audit type. The technical properties of the entry fields are defined in Customizing. This allows you to cover different audit types during an implementation.

Further notes

You can find technical details about the individual methods in the interface documentation for the interface: IF_EX_PLM_AUDIT_OBJECT.

Methods

- Consistency Check of Audit Object (->Error Log)
- Input Help Using F4 for the Audit Object
- Display Audit Object (Table) in the Audit Overview
- Read Description of Audit Object
- Display of the Audit Object
- Conversion of User Entries to Internal Data Format

BAdI: Calculation/Valuation of an Investigation (Audit, FMEA...)

Use

This Business Add-In is used in the *Audit Management* component and in derived components such as the *Failure Mode and Effects Analysis* (FMEA).

This BAdI contains the methods for the calculating and valuating the audit question list and the audit as a whole, or the general valuation of an investigation.

When implementing this BAdI, you must specify a filter type before the implementation of the methods can take place. This filter type corresponds to the field *Procedure* in the question list, in the audit question list, and in the audit, or in the investigation and its list items.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.

Note: You can also create an implementation for an Add-In and not activate it until later.
If you want to do this, do not carry out the following step:

8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.
   Example

To display sample code, choose *Goto -> **Sample Code** -> Display*.

See also:

Interface

IF_EX_PLM_AUDIT_CALCULATE.

Methods

- Calculation of Audit Question List
- Calculation of the Audit
- Read Various Settings

- Display of Valuation Information
- Check Whether Valuation Information Can Be Displayed

##

BAdIs Within the Audit Application

Maintenance of Help Links

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to implement enhancements within *Audit Management* (transaction code PLMD_AUDIT).

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods

Getting the URL for Initial Screen

Object-Specific Processing of Help Request

##
#

Enhancement for Search Helps

Use

This Business Add-In (BAdI) is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to change the input help (F4 help) values for a number of fields.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.

Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:

8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display the sample code, choose *Goto*--> *Sample Code* --> *Display*.

See also:

Methods

- Read Values for Grouping
- Read Values for 'Search Field'
-

Definition of Text Types for Each Audit Component

Use

This Business Add-In (BAdI) is used in the *Audit Management* (*CA-AUD*) component.

You use this BAdI to set long text processing for the different audit components.

Requirements

The method GET_TEXT_IDS is only relevant outside of a CRM system.

If you are working in a CRM system, you edit the activity *Define Text Objects and Text Types* in the Implementation Guide (IMG) instead of the method GET_TEXT_IDS. The text types used in audit management are defined using the text object CGPL_TEXT in the CRM standard system.

Standard settings

The following text types are supplied in the standard system:

Text determination procedure/Object types	Description
AQN	Audit Management: Audit question
AUO	Audit Management: Audit
AUP	Audit Management: Audit plan
COR	Audit Management: Corrective actions
QUE	Audit Management: Question
QUN	Audit Management: Question lists

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Copy the standard implementation and change the text types for each audit component.

Example

To display the sample code, choose *Goto --> Sample Code --> Display*.

See also:

Method:

Edit Text Type

Returns the Text IDs for the XML Transfer of Questions

Returns the Text IDs for the XML Transfer of Audit Questions

Provide Signature for Audit

Use

This Business Add-In is used in the *Audit Management* (*CA-AUD*) component.

You can use this BAdI to influence the signing of an audit. For example, it is possible to perform a digital signature.

Standard settings

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

To display sample code, choose *Goto -> Sample Code -> Display*.

Description

See also:

Methods

Signing of Audit

Display of Signatures

Display Flexibilization of "Last Associated Audit"

Use

This Business Add-In (BAdI) is used in the *Audit Management (CA-Audit)* component.

The BAdI runs when you choose the pushbutton *Last Audit*.

Requirements

This Business Add-In is not active in the standard system.

Activities

After you call the IMG activity, the system displays a dialog box where you enter a name for the implementation.

If implementations of this Business Add-In have already been created, the system displays them in a dialog box. You then choose one of them by choosing ***Create***, and continue as follows:

1. In the dialog box, enter a name for the implementation of the Add-In and choose ***Create***. The system displays the initial screen for creating Business Add-In implementations.
2. On this screen, enter a short description for you implementation in the ***Implementation Short Text*** field.
3. If you choose the ***Interface*** tab, you will notice that the system has filled in the ***Name of the Implementing Class*** field automatically, by assigning a class name based on the name of your implementation.
4. Save your entries and assign the Add-In to a package.
5. To edit a method, double-click its name.
6. Enter your implementation code between the `method <Interface Name>~<Name of Method>.` and `endmethod.` statements.
7. Save and activate your code. Navigate back to the ***Change Implementation*** screen.
   Note: You can also create an implementation for an Add-In and not activate it until later. If you want to do this, do not carry out the following step:
8. Choose ***Activate***.
   When the application program is executed, the system carries out the code in the method you wrote.

Example

The following process has been implemented in the sample code:

You want to select a specific audit from all the relevant audits.

To display sample code, choose *Goto -> Sample Code -> Display*.

See also:

Methods

- SELECTION_CRITERIONS
- CHOOSE_LAST_AUDIT

BAdI: Restriction of Partner Roles for Audit Components

Use

This Business Add-In (BAdI) is used in the *Audit Management* (CA-AUD) component and in other derived components, such as Failure Mode and Effects Analysis (FMEA).

You can use this BAdI to restrict the partner roles that can be used for the individual object types.

Standard settings

For more information about the standard settings (filters, single or multiple use), see the *Enhancement Spot Element Definitions* tab in the BAdI Builder (transaction SE1).

BAdI Implementations

PLM_AUD_ROLES_RSTR_DEF
The default implementation is intended for use in conjunction with the *Restrict Roles* indicator in the Customizing activity *Cross-Application Components -> Audit Management -> Control -> Audit Components: Properties, Forms*.

The default implementation for the BAdI is implemented via the class CL_IM_PLM_AUD_ROLES_RSTR_DEF (Class Documentation).

With the default implementation of the BAdI, you can restrict the roles for an audit type or an FMEA type:

- You set the *Restrict Roles* indicator for an audit or an FMEA in Customizing under *Cross-Application Components -> Audit Management -> Control -> Audit Components: Properties, Forms* or under *Quality Management -> Quality Planning -> Failure Mode and Effects Analysis (FMEA) -> Control Data -> Data Objects: Properties, Forms*.
- In Customizing, you specify the permitted roles for an audit type or FMEA type under *Cross-Application Components -> Audit Management -> Audit Definition -> Audit Type* in the *Settings for Role* folder for an audit, or under *Quality Management -> Quality Planning -> Failure Mode and Effects Analysis (FMEA) -> FMEA Definition -> FMEA Types* in the *Settings for Roles* folder for an FMEA.
- If you then change existing audits or FMEAs whose roles deviate from the specified roles, the system issues a warning.
- If you specify differing partner roles when you create audits and FMEAs, the system displays an error message.

If you want to restrict the partner roles for audit plans, corrective and preventive actions, and question lists, you must create your own customer-specific implementation of the BAdI.

Activities

Additional Information

BAdI method documentation

- GET_VALID_ROLES_FOR_ASSESSMENT
- GET_VALID_ROLES_FOR_QUEST_LIST
- GET_VALID_ROLES_FOR_ACTION

- GET_VALID_ROLES_FOR_AUDIT_PLAN

For more information about implementing BAdIs as part of the Enhancement Framework, see SAP Library for SAP NetWeaver Platform on SAP Help Portal at http://help.sap.com/nw_platform. Choose a release and then Application Help. In SAP Library, choose SAP NetWeaver Library: *Function-Oriented View -> Application Server -> Application Server ABAP -> Application Development on AS ABAP -> ABAP Customer Development -> Enhancement Framework*.

BAdIs for Archiving

Archiving Object PLM_AUD: Checks for Add-On-Specific Data

Use

The Business Add-In (BAdI) ARC_PLM_AUD_CHECK is used in the Audit Management (CA-AUD) component the program PLM_AUDIT_ARC_AUD_CHECK (preprocessing program for archiving movement data related to audit management). It also allows additional checks of archivability criteria via the method Check Archivability of Transaction Data (CHECK).

For more information on how to implement the BAdI methods, see SAP Note 673030.

Requirements

None.

Standard settings

The Business Add-In can be implemented more than once.

Archiving Object PLM_AUD: Archiving of Add-On-Specific Data

Use

The Business Add-In (BAdI) ARC_PLM_AUD_WRITE is used in the *Audit Management (CA-AUD)* component, in the methods WRITE_ADDONS and DELETE_ADDONS of the class CL_PLM_AUDIT_COMP_ARCHIVING.

These methods are called by the programs PLM_AUDIT_ARC_AUD_WRITE (write program for archiving of movement data in Audit Management) and PLM_AUDIT_ARC_AUD_DELETE (deletion program for archiving of movement data in Audit Management) via the methods WRITE and DELETE of the same classes. The methods allow you to write additional data to the archive and to delete this data from the database.

The BAdI has the following methods:

- Deletes Additional Data from Database (DELETE)
- Reads Additional Data from Database(PREPARE_WRITE)

- Writes Additional Data to Archive (WRITE)

For more information about the implementation of the BAdI methods, see SAP Note 673030.

Requirements

None.

Standard settings

The Business Add-In can be implemented more than once.

Archiving Object PLM_QUM: Checks for Add-On-Specific Data

Use

The Business Add-In (BAdI) ARC_PLM_QUM_CHECK is used in the *Audit Management (CA-AUD)* component in the program PLM_AUDIT_ARC_QUM_CHECK (preprocessing program for archiving master data in Audit Management) and allows you to perform additional checks of your own archivability criteria via the method Check Archivability of Master Data(CHECK).

For more information on the implementation of the BAdI methods, see SAP Note 673030.

Requirements

None.

Standard settings

The Business Add-In can be implemented more than once.

Archiving Object PLM_QUM: Archiving of Add-On-Specific Data

Use

The Business Add-In (BAdI) ARC_PLM_QUM_WRITE is used in the Audit Management (CA-AUD) component, in the methods WRITE_ADDONS and DELETE_ADDONS of class CL_PLM_AUDIT_COMP_ARCHIVING.

These methods are called by the programs PLM_AUDIT_ARC_QUM_WRITE (write program for archiving of master data in Audit Management) and PLM_AUDIT_ARC_QUM_DELETE (deletion

program for archiving of master data in Audit Management) via the methods WRITE and DELETE of the same class. The methods allow you to write additional data to the archive and to delete this data from the database.

The BAdI has the following methods:

Deletes Additional Data from Database (DELETE)

Reads Additional Data from Database (PREPARE_WRITE)

Writes Additional Data to Archive (WRITE)

For more information on the implementation of BAdI methods, see SAP Note 673030.

Requirements

None.

Standard settings

The Business Add-In can be implemented more than once.