SAP Tree and Tree Model (BC-CI)

HELP.BCCITREE

Release 4.6C

Copyright

Icons

Icon	Meaning
⚠	Caution
💬	Example
➡	Note
🧭	Recommendation
SYN	Syntax
💡	Tip

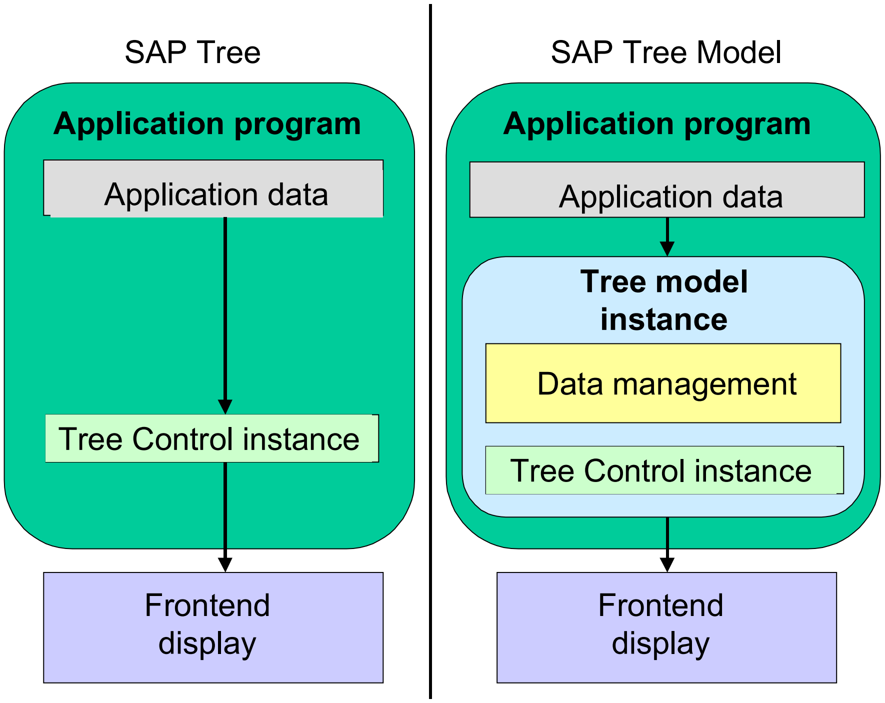
Content

SAP Tree and Tree Model (BC-CI)

Purpose

SAP Tree and SAP Tree Model are techniques based on SAP's Control Framework that you can use to display hierarchically-arranged data in tree format. SAP Tree, introduced in Release 4.6A, can be used to display data at the frontend. The SAP Tree Model, new in Release 4.6C, also allows you to administer the data within the control instance.

The graphic illustrates the difference: The SAP Tree receives application data from the program and passes it to its corresponding frontend component, either directly, or at the next synchronization. However, it cannot hold any data itself, which means that to find out attributes of a node or item other than its key or name, you must program the request yourself on the basis of the node key. The SAP Tree Model, on the other hand, incorporates a data management part and also encapsulates a normal tree control instance. All of the data that is passed to the tree is held within the Tree Model instance as well as being sent to the frontend. Consequently, actions like searching within the tree take place within the ABAP program, and do not require time-consuming network communication.



Implementation Considerations

When deciding whether to use the SAP Tree or the SAP Tree Model, you should remember that the SAP Tree Model provides some useful functions that are not available in the SAP Tree. For example, the SAP Tree Model allows you to:

- Check node keys before you send them to the frontend (to ensure that there are no duplicates)
- Search within the tree

- Print the tree

The SAP Tree Model also provides

- Automatic flush handling
- Automatic handling of node transfer to the frontend

To use the SAP Tree you must have a SAPgui with Release 4.6A or higher. To use the SAP Tree Model, you must have a SAPgui with Release 4.6C or higher.

Constraints

Certain features of the SAP Tree and SAP Tree Model are not available under SAPGUI for HTML. For further information, refer to The Tree Controls in SAPGUI for HTML [Page 14].

The Tree Controls in SAPGUI for HTML

Part of SAP's GUI strategy has been to introduce the SAPGUI for HTML, in which it is possible to use R/3 transactions within a web browser. However, certain functions of the Tree and Tree Model controls are either not available in this environment, or have different behavior from the SAPGUI for Windows environment.

Restrictions in the Simple Tree and Simple Tree Model

- You cannot set and retrieve the top node of the tree control display. The top node of the display cannot be retained between browser requests
- Drag and drop is not available
- Context menus are not available for nodes

Restrictions in the Column Tree and Column Tree Model

All of the restrictions of the Simple Tree and Simple Tree Model apply, along with the following:

- You cannot set or change the column width. The column widths are automatically fitted to the maximum text width in the column
- You cannot scroll the hierarchy part of the tree separately

Restrictions in the List Tree and List Tree Model

All of the restrictions of the Simple Tree and Simple Tree Model apply, along with the following:

- Colors and formatting have not yet been implemented

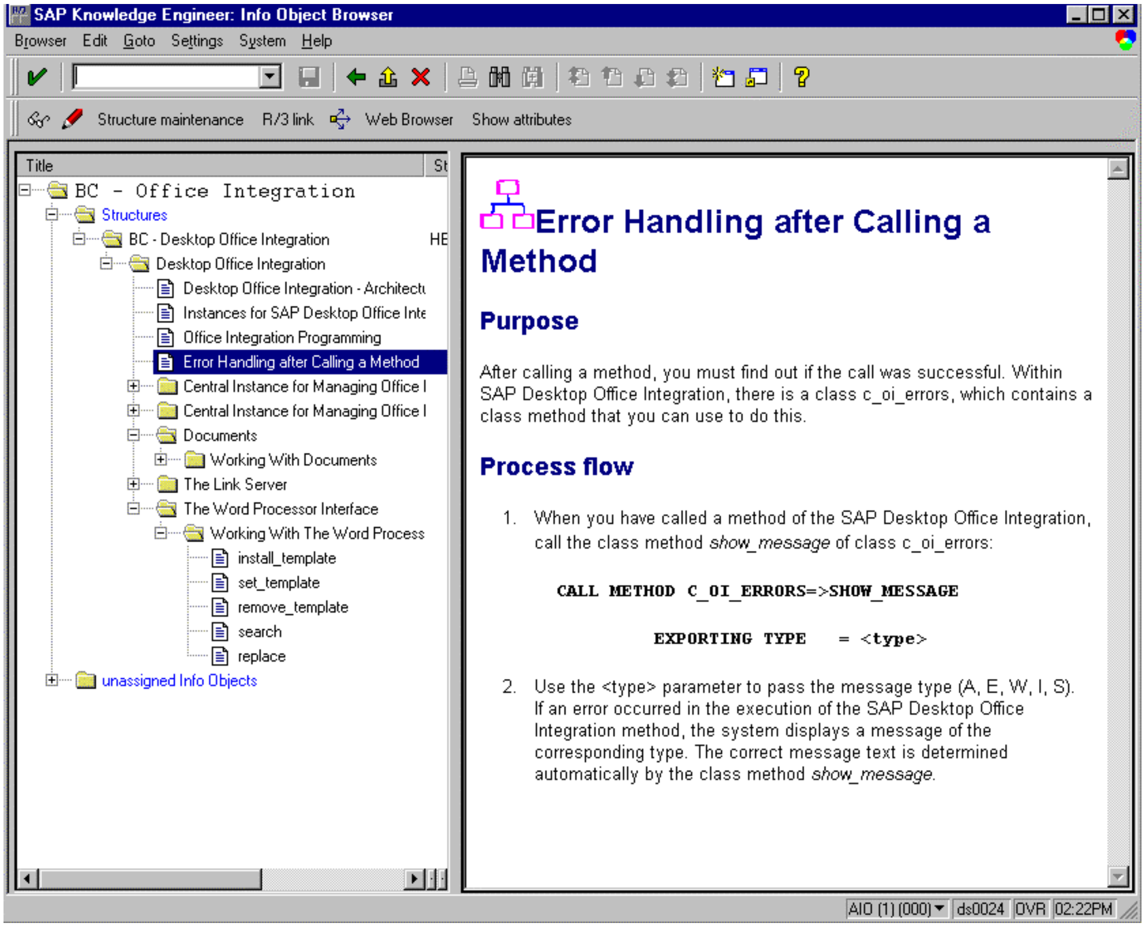
SAP Tree

Purpose

SAP Tree is a control that allows you to display tree structures on a screen. It has been developed by SAP, and while it fulfills the basic requirements of a tree control, it has not been adapted for individual applications.

The R/3 System contains the following example programs: **`SAPCOLUMN_TREE_CONTROL_DEMO`**, **`SAPSIMPLE_TREE_CONTROL_DEMO`**, and **`SAPTLIST_TREE_CONTROL_DEMO`**.

The following graphic provides an example. The R/3 window contains both a SAP Tree (left-hand side) and a SAP HTML Viewer (right-hand side):


SAP Knowledge Engineer: Info Object Browser
Browser Edit Goto Settings System Help
Structure maintenance R/3 link Web Browser Show attributes
Title
St
BC - Office Integration
Structures
BC - Desktop Office Integration
HE
Desktop Office Integration
Desktop Office Integration - Architectu
Instances for SAP Desktop Office Inte
Office Integration Programming
Error Handling after Calling a Method
Central Instance for Managing Office I
Central Instance for Managing Office I
Documents
Working With Documents
The Link Server
The Word Processor Interface
Working With The Word Process
install_template
set_template
remove_template
search
replace
unassigned Info Objects
AIO (1) (000)
ds0024
OVR
02:22PM

Error Handling after Calling a Method

Purpose

After calling a method, you must find out if the call was successful. Within SAP Desktop Office Integration, there is a class c_oi_errors, which contains a class method that you can use to do this.

Process flow

1. When you have called a method of the SAP Desktop Office Integration, call the class method *show_message* of class c_oi_errors:

```
CALL METHOD C_OI_ERRORS=>SHOW_MESSAGE

        EXPORTING TYPE    = <type>
```

2. Use the <type> parameter to pass the message type (A, E, W, I, S). If an error occurred in the execution of the SAP Desktop Office Integration method, the system displays a message of the corresponding type. The correct message text is determined automatically by the class method *show_message*.

Features

There are three different versions [Page 18] of the SAP Tree.

- Simple tree structure: A simple tree with a single text entry for each node.

- List structure: Each node may have more than one entry. The entries are displayed from left to right.
- Column structure: Tree structure with freely-definable columns.

Programming the SAP Tree

Data Handling in the ABAP Program:

When you program the SAP Tree, you send it the data you want to display, along with the necessary administration and formatting information.

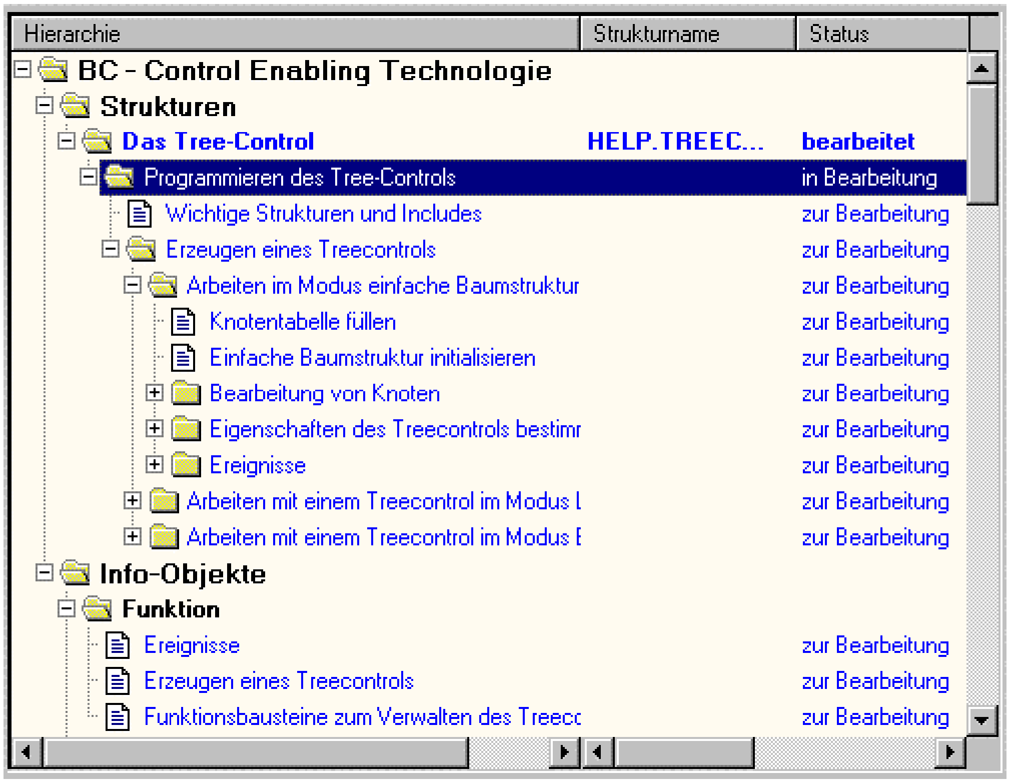
An instance of the SAP Tree has no data of its own. Instead, you use it to transfer data to and from the SAP Tree at the frontend. You must maintain the tree data structure in your application.

This is particularly important in event handling, since the tree control only provides administrative data when an event is triggered (for example, the name of the node on which the event was triggered).

Important Elements of the SAP Tree

Nodes with subordinate nodes are called branches. Nodes without subordinate nodes are called leaves.

Each folder that is not empty has a plus or minus sign next to it, indicating that you can expand or collapse it respectively . You can also define a picture for each node, which is displayed before the node itself.



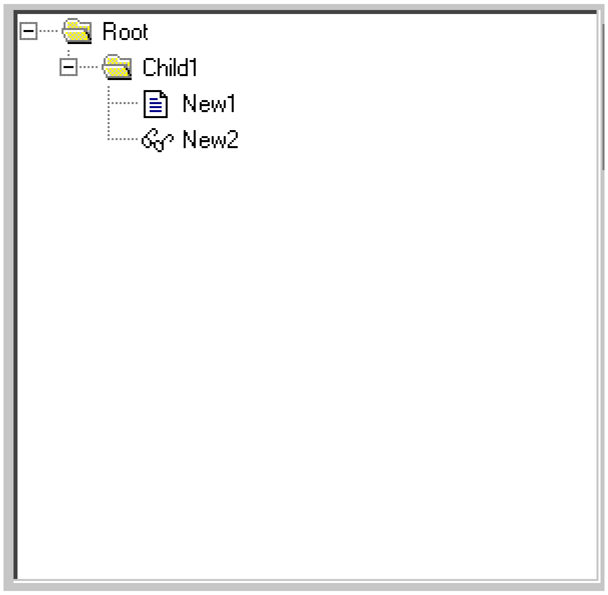
```
Hierarchie                                          Strukturname    Status
⊟ BC - Control Enabling Technologie
  ⊟ Strukturen
    ⊟ Das Tree-Control                              HELP.TREEC...   bearbeitet
      ⊟ Programmieren des Tree-Controls                             in Bearbeitung
          Wichtige Strukturen und Includes                          zur Bearbeitung
        ⊟ Erzeugen eines Treecontrols                               zur Bearbeitung
          ⊟ Arbeiten im Modus einfache Baumstruktur                 zur Bearbeitung
              Knotentabelle füllen                                  zur Bearbeitung
              Einfache Baumstruktur initialisieren                  zur Bearbeitung
            ⊞ Bearbeitung von Knoten                                zur Bearbeitung
            ⊞ Eigenschaften des Treecontrols bestimr                zur Bearbeitung
            ⊞ Ereignisse                                            zur Bearbeitung
          ⊞ Arbeiten mit einem Treecontrol im Modus L               zur Bearbeitung
          ⊞ Arbeiten mit einem Treecontrol im Modus E               zur Bearbeitung
  ⊟ Info-Objekte
    ⊟ Funktion
        Ereignisse                                                  zur Bearbeitung
        Erzeugen eines Treecontrols                                 zur Bearbeitung
        Funktionsbausteine zum Verwalten des Treecc                 zur Bearbeitung
```

Overview of SAP Tree Classes

Simple Tree

The class **CL_GUI_SIMPLE_TREE** is the ABAP Objects wrapper for the simple tree.

Example program: **SAPSIMPLE_TREE_CONTROL_DEMO**:



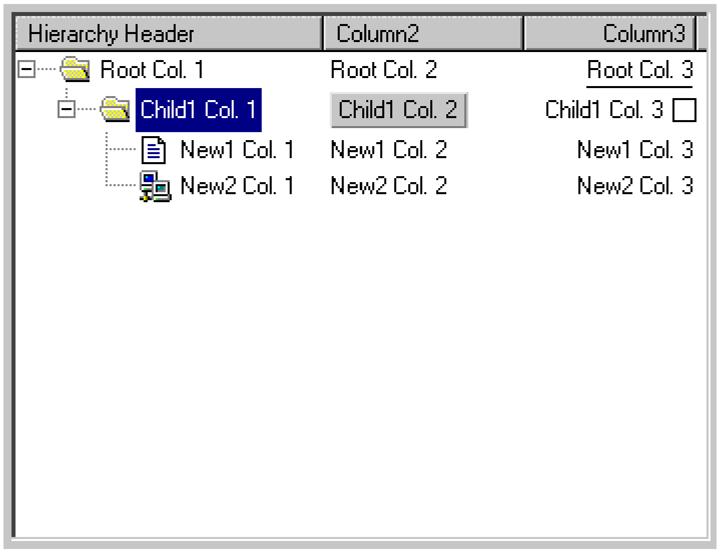
Attributes

- A node consists of a folder or leaf symbol and a text.
- You cannot use checkboxes or additional icons.
- You can only have one text for each node.
- There is no heading.

Column Tree

The class **CL_GUI_COLUMN_TREE** is the ABAP Objects wrapper for the column tree.

Example program: **SAPCOLUMN_TREE_CONTROL_DEMO**:



Hierarchy Header	Column2	Column3
Root Col. 1	Root Col. 2	Root Col. 3
Child1 Col. 1	Child1 Col. 2	Child1 Col. 3 □
New1 Col. 1	New1 Col. 2	New1 Col. 3
New2 Col. 1	New2 Col. 2	New2 Col. 3

Attributes

- A node consists of a folder or leaf symbol and a range of items.
- The entries of a node are arranged in columns.

  In the example, the tree has three columns with the logical names 'Column1', 'Column2', and 'Column3'. The topmost node has an entry in each of these columns:

  'Root Col. 1' in column 'Column1'

  'Root Col. 2' in column 'Column2'

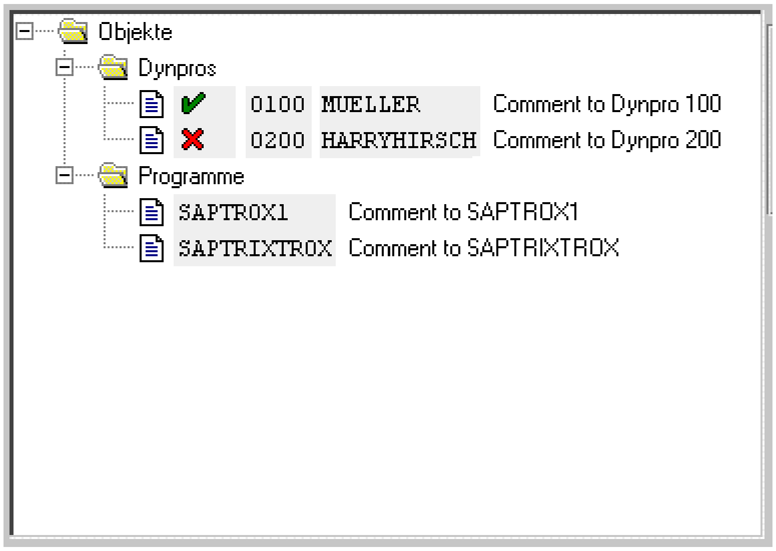
  'Root Col. 3' in column 'Column3'

- A COLUMN_TREE can contain two kinds of columns:
  - Columns in the hierarchy area: These columns are below the hierarchy heading. The hierarchy heading is the first heading from the left in the SAP Tree (in the example, 'Hierarchy Header'). There is normally only one column in the hierarchy area. In the example, it is the column with the name 'Column1', containing the entries 'Root Col.1', 'Child1 Col. 1' and so on.
  - Columns outside the hierarchy area: These columns have their own heading. The example contains two columns outside the hierarchy area, with the headings 'Column2' and 'Column3'.
- Columns can have the following kinds of entries:
  - Text: Text, with optional icon
  - Checkbox: Checkbox with optional icon and text.
  - Pushbutton: Pushbutton with text and icon.
  - Link: Like text, but additionally, an event is triggered when the user clicks the link.

List Tree

The class **`CL_GUI_LIST_TREE`** is the ABAP Objects wrapper for the column tree.

Example program: **`SAPTLIST_TREE_CONTROL_DEMO`**:



```
Objekte
  Dynpros
     ✓  0100 MUELLER      Comment to Dynpro 100
     ✗  0200 HARRYHIRSCH  Comment to Dynpro 200
  Programme
     SAPTROX1     Comment to SAPTROX1
     SAPTRIXTROX  Comment to SAPTRIXTROX
```

Attributes

- A node consists of a folder or leaf symbol and entries.
- The entries are displayed from left to right.

  Structure of the first three nodes in the example:

  The topmost node has a single entry ("objects"). Proportional font is set for this entry. Additionally, the "automatic width" is set. This means that the width of the entry is adjusted to fit the contents (in this case, the string "objects").

  The second node from the top has the same construction as the first: An entry with the text "Screens".

  The third node from the top has four entries:

  A tick icon, four characters wide.

  0100, not in proportional font, four characters wide.

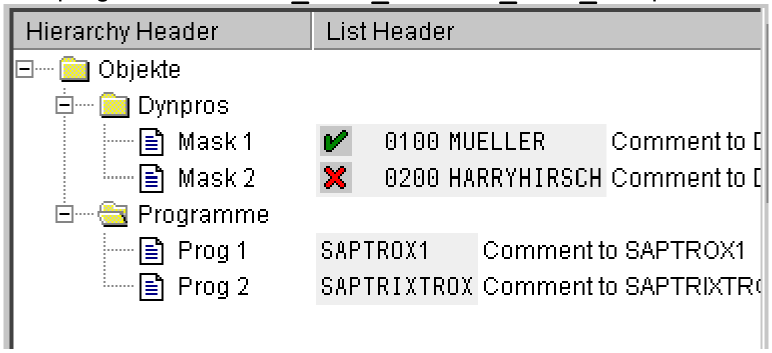
  MUELLER, not in proportional font, 11 characters wide.

  Comment for screen 100, proportional font, automatic width.

- Using non-proportional fonts and a fixed display width allows you to display data in tabular format, as in the example.
- Columns can have the following kinds of entries:
  - Text: Text, with optional icon
  - Checkbox: Checkbox with optional icon and text.

  - Pushbutton: Pushbutton with text and icon.
  - Link: Like text, but additionally, an event is triggered when the user clicks the link.

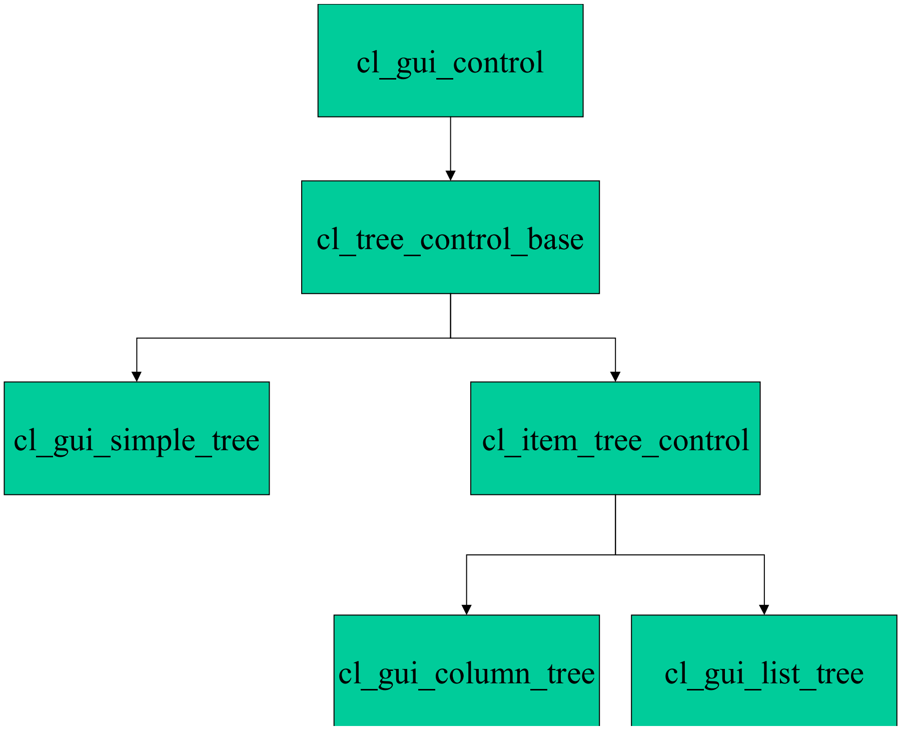
- There is a hierarchy heading and a list heading, under which all entries can be grouped. The program **SAPTLIST_TREE_CONTROL_DEMO_HDR** provides an example:



Hierarchy Header	List Header		
Objekte			
Dynpros			
Mask 1	✓	0100 MUELLER	Comment to D
Mask 2	✗	0200 HARRYHIRSCH	Comment to D
Programme			
Prog 1	SAPTROX1	Comment to SAPTROX1	
Prog 2	SAPTRIXTROX	Comment to SAPTRIXTR	

The Inheritance Hierarchy

The lines in the diagram indicate the inheritance relationship.



All control classes inherit from the class **`cl_gui_control`**.

The interface of a control consists of the public methods of its wrapper class and of the superclasses of the wrapper class.

Interface of the class **`cl_gui_column_tree`**:

Public methods of the class **`cl_gui_column_tree`**

Public methods of the class **`cl_item_tree_control`**

Public methods of the class **`cl_tree_control_base`**

Public methods of the class **`cl_gui_control`**

Functions of the Classes

cl_tree_control_base	Methods common to all SAP Trees (simple tree, list tree, column tree). For example - setting the expanded node.
cl_gui_simple_tree	Simple Tree
cl_item_tree_control	Methods common to the list tree and column tree. For example, setting the text for an item.
cl_gui_column_tree	Column Tree

cl_gui_list_tree	List Tree

Finding Errors

The majority of errors in control programming occur when you synchronize the automation queue [Ext.]. Synchronization occurs either explicitly, using the method CL_GUI_CFW=>FLUSH [Page 474], or implicitly after the last PBO module has finished.

If the error occurs in an explicit synchronization, the method **CL_GUI_CFW=>FLUSH** triggers the exception **CNTL_ERROR**. If the error occurs in an implicit synchronization, a short dump occurs. You can avoid the short dump by handling special events of the Control Framework.

The exception **CNTL_ERROR** only indicates that an unspecified method call to a control at the frontend was unsuccessful. You then need to find out which control at the frontend has triggered the exception and why. You can do this using the Debugger:

1. Run the program again in the Debugger.
2. Go into the settings in the Debugger and select the option *Automation Controller: Always process requests synchronously*.
   When you set this option, the automation queue is synchronized after each method call.
3. Step through the individual method calls. Note that **SY-SUBRC** is only set after the method that triggers the exception if you handle the exceptions in your application program. Otherwise, another short dump occurs.
4. Identify the error in the method call.

💡

If an error occurs, you should first run the example programs for the corresponding control wrapper. If an error also occurs in these programs, the problem is due to your local SAPgui installation.

⚠

Once **CNTL_ERROR** has been triggered, you should no longer work with the controls. Remember above all that method calls that come after the error in the automation queue will not be processed.

If the error occurred in the first automation queue synchronization, the automation controller may no longer be active. This results in all subsequent control calls ending with a **CNTL_ERROR**.

Important Notes

If you transfer too many nodes to the SAP Tree within a single PBO/PAI cycle, a timeout may occur. The Incremental Tree Construction [Page 31] section explains how you can minimize the number of nodes transferred at any one time.

If you want to change a large number of components (for example, 20 texts), you should use a method with a table interface (update_nodes_and_items [Page 175] or update_nodes [Page 185]) instead of calling a single method 20 times. This also applies to other operations, such as expanding nodes.

Within a PBO/PAI cycle, you should not repeatedly call methods with table interfaces. For example, instead of calling one of the ADD_NODES_… methods 20 times with five nodes in each call, it is better to call it once with all 100 nodes.

The exceptions of the SAP Tree methods do not set messages.

You must never ignore exceptions of the SAP Tree methods or flush calls. If an error occurs, the automation queue processing is terminated. This affects all of the controls in the same internal session. Once an error has occurred, the internal session affected may no longer work with controls.

The SAP Tree is not suitable for displaying non-hierarchical lists, since all root nodes must always be transferred to the control. Consequently, long lists cause performance problems.

Finding Errors

If an error suddenly occurs in a program that previously worked correctly, you should test the SAP Tree examples to see if they still work.

The majority of errors in control programming occur when you call the flush [Page 474] method. The exception CNTL_ERROR only indicates that an unspecified error has occurred in a control at the frontend. The error does not actually have to have occurred in the SAP Tree - it can be triggered by any control.

To find the error, restart the program in the Debugger. Go into the *Settings* screen in the Debugger. Select the option *Automation Controller:* Always process requests synchronously (see Automation Queue Services [Ext.]). The individual SAP Tree methods will now return more precise information about where the error occurred.

Example Programs

Your system contains the following example programs for the SAP Tree:

Example program	**Theme**
SAPSIMPLE_TREE_CONTROL_DEMO	Example of a simple tree
SAPTLIST_TREE_CONTROL_DEMO	Example of a list tree
SAPCOLUMN_TREE_CONTROL_DEMO	Example of a column tree
SAPSIMPLE_TREE_CONTEXT_MEN_DEM	Example of context menus
SAPTLIST_TREE_CONTROL_DEMO_HDR	Example of a context menu on headings in a SAP Tree
SAPSIMPLE_TREE_DRAG_DROP_DEMO	Example of drag and drop
RSDEMO_DRAG_DROP_TREE_MULTI	Example of drag and drop with multiple selection
RSDEMO_DRAG_DROP_EDIT_TREE	Example of drag and drop between a SAP Tree and a SAP Textedit

Using Controls in a WAN

When you use controls in your programs, you place an extra load on the communication channel between the frontend and backend. In a LAN, and particularly in a WAN environment, this can be a critical factor.

The problem is alleviated somewhat by buffering mechanisms (see also Automation Queue [Ext.]). Use these points as a guideline to using controls in a WAN.

The documentation for the individual controls also contains more specific notes about using that control in a WAN.

Using CL_GUI_CFW=>FLUSH

The method CL_GUI_CFW=>FLUSH [Page 474] synchronizes the automation queue and the ABAP variables in it. Calling it often generates a synchronous RFC call from the application server to the frontend. To optimize the performance of your application, you should call this method as little as possible.

It is often a good idea to read all control attributes in a single automation queue (for example, at the beginning of the PAI) and retrieve them in a single synchronization. You should, in particular, do this when you read attributes that are not necessary in your event handlers or the PAI/PBO cycle.

You do not need to include a "safety flush" at the end of the PBO to ensure that all method calls are transported to the frontend. A flush at the end of the PBO is guaranteed. Consequently, you cannot construct an automation queue spread over several screens.

There is no guarantee that an automation queue will be sent when you call `CL_GUI_CFW=>FLUSH`. The queue recognizes whether it contains any return values. If this is not the case, it is not sent.

If you have a queue with no return values, and want to ensure that it is synchronized, you can use the Control Framework method CL_GUI_CFW=>UPDATE_VIEW [Page 477]. You should only use this method if you absolutely need to update the GUI. For example, you might have a long-running application in which you want to provide the user with regular updates on the status of an action.

After you have read the attributes of a control, the contents of the corresponding ABAP variables are not guaranteed until after the next flush. The contents of the ABAP variables remain undefined until this call occurs. In the future, there will be cases in which this flush is unnecessary. They will be recognized by the automation queue and the corresponding flush call will be ignored.

Creating Controls and Passing Data

Creating controls and passing data to them is normally a one-off procedure, which in comparison to using normal screen elements can be very runtime-intensive. You should therefore not use any unnecessary controls, or pass unnecessary data to the controls that you are using.

A typical example is a tabstrip control with several tab pages. If the pages contain controls, you should consider using application server scrolling instead of local scrolling, and not loading the controls until the corresponding page is activated by the user. The same applies to passing data to the controls on tab pages.

If you want to differentiate between LAN and WAN environments when you pass data to a control, you can use the function module **`SAPGUI_GET_WANFLAG`**. In some applications, you may need to pass different amounts of data or use a complete fallback in a WAN application. The environment affects, for example, the number of same-level nodes that you can transfer to a tree control without having to introduce artificial intermediate levels.

Unlike screen elements, controls only have to be created and filled with data once. From a performance point of view, this means that they become more profitable the longer they exist. In applications that are called repeatedly, and therefore initialized repeatedly, controls can have a negative effect on performance. In applications that use the same screen for a long time, on the other hand, you may find that using controls results in improved performance.

You can always use the performance tools [Ext.] to check the advantages and disadvantages in terms of network load that using a control brings.

Storing Documents, Picture, and Other Data

Release 4.6A sees the introduction of a frontend cache for accessing documents from the Business Document Service (BDS). You are strongly recommended to store desktop documents, images, and other data in the BDS and not in the R/3 database. Documents from the BDS can be cached at the frontend, and therefore only have to be loaded over the network once.

Special Considerations for the SAP Tree

In addition to the considerations that apply to all controls, you should note the following:

Wherever possible when you use the SAP Tree, you should avoid loading child nodes [Page 31] until the user expands the parent node. If a hierarchy level has a large number of nodes, you should insert artificial intermediate levels. This also gives the user a better overview of your tree structure.

Filling a tree control with a deep hierarchy structure can be a runtime-intensive operation. This problem is not restricted to use in a WAN - it can also occur in a LAN environment. As well as the large amount of data that has to be transferred for a large hierarchy, considerable runtime is also expended inserting the data into the control. Running under a 200 MHz processor, the control can insert around 700 nodes per second into a simple tree (no additional columns).

Since the SAP tree uses keys instead of line and column numbers, and there is no general data model, it is impossible to provide a general solution. When you use the SAP Tree, you must ensure that your programming method results in acceptable performance.

There are three ways of avoiding the problem:

Loading Child Nodes on Demand

See Incremental Tree Construction [Page 31].

Adding Artificial Intermediate Levels to the Hierarchy

If a node has a large number of child nodes, transferring even only the child nodes of that one node can cause performance problems. Furthermore, if the list of child nodes for a single node extends over several pages, the tree becomes less easily readable for the user. _

If you have a node with several child nodes, you can divide them up by using artificial intermediate hierarchy levels. From a technical point of view, a sensible number of same-level nodes is around 500 in a LAN, and around 100 in a WAN.

```
Product
    |____ Vendor1
    |____ Vendor2
    |____ Vendor3
       ...
    |____ Vendor1000
```

You could alleviate this problem by inserting intermediate hierarchy levels, whose child nodes are also only loaded on demand, as follows:

```
Product
    |_____| Vendor1 – Vendor100
    |             |_____ Vendor1
    |             |_____ Vendor2
    |             |_____ ...
    |             |_____ Vendor100
    |
    |_____ Vendor101 – Vendor200
and so on.
```

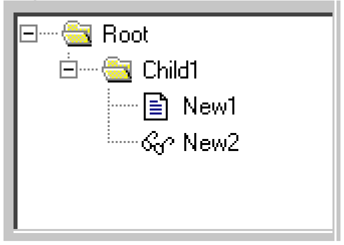
Explorer-Type Structure

Instead of displaying the leaves of the tree in the tree itself, you could display them in a table control to the right of the tree. In this case, only the folders are displayed in the tree. The leaves, which form a large part of the data, no longer have to be transferred.

Incremental Tree Construction

Transferring node informatino to the SAP Tree at the frontend is a critical operation in performance terms. If you have a large tree with more than 500 nodes, you should not transfer the child nodes of a particular node to the frontend until the user actually expands that node.

In the following structure, you should only add the nodes Root and Child1 into the SAP Tree. The child nodes of Child1 should not be transferred until the user actually expands the node.



To do this, you must implement the following steps:

- For node Child1, set the field **`EXPANDER = 'X'`** in the node structure (**`TREEV_NODE`**). By doing this, you ensure that the event **`EXPAND_NO_CHILDREN`** is triggered when the user tries to expand this node.
- Register the event **`EXPAND_NO_CHILDREN`**.
- In the event handler method, include a runtime that transfers the child nodes of the expanded node to the SAP Tree.

An example of this is provided in the program **`SAPSIMPLE_TREE_CONTROL_DEMO`**.

The Simple Tree

Definition

You crate a simple tree with reference to the class **cl_gui_simple_tree**:

DATA simple_tree TYPE REF TO cl_gui_simple_tree.

According to the inheritance hierarchy, you can now access the methods of the following classes:

- **cl_gui_object** and **cl_gui_control** (see Methods of the ABAP Objects Control Framework [Page 471]).
- **cl_gui_tree_control_base** (see Methods of the Class CL_TREE_CONTROL_BASE [Page 119]).
- **cl_gui_simple_tree** (see Methods of the Class CL_GUI_SIMPLE_TREE [Page 179]).

Use

The program **sapsimple_tree_control_demo** demonstrates how to use the simple tree.

For details of the attributes of the simple tree, refer to the Overview of Tree Classes [Page 18].

Creating a Control: SAP Picture Example

Prerequisites

The following process applies to all SAP custom controls. The programming examples use the SAP Picture Control. However, to apply the example to other controls, you would only have to change the name of the control class.

The example also assumes that you are using the custom control in a Custom Container. The SAP Container documentation contains details of further scenarios.

Process Flow

Create the Instance

1. Define a reference variable for the Custom Container in which you want to place the custom control (**see** SAP Container [Ext.]).

```
DATA container TYPE REF TO cl_gui_custom_container.
```

2. Define a reference variable for the SAP Picture:

```
DATA picture TYPE REF TO cl_gui_picture.
```

3. Create the Custom Container. You must already have created the area `'CUSTOM'` for the Custom Container in the Screen Painter. When you create the container, you must also specify its lifetime [Ext.] (see constructor [Ext.]).

```
CREATE OBJECT container
   EXPORTING container_name = 'CUSTOM'
          lifetime       = lifetime.
```

4. Create the SAP Picture Control. You can also specify a lifetime for the SAP Picture, but it must not be longer than that of its container.

```
CREATE OBJECT picture
    EXPORTING parent  = container
          lifetime = lifetime.
```

Register the Events

5. There are three steps: Registering the events with the Control Framework, defining a handler method, and registering the hander method. These steps are explained under Registering and Processing Events [Page 99].

Use the Control

6. These steps are control-specific and therefore not described here.

Destroy the Control

The lifetime management [Ext.] is normally responsible for destroying any controls you use. However, the following two steps allow you to destroy the control yourself:

7. Use the method free [Page 480] to destroy the Custom Control at the frontend. If you no longer need the control container, release it as well:

Creating a Control: SAP Picture Example

```
CALL METHOD picture->free
    EXCEPTIONS cntl_error        = 1
           cntl_system_error = 2.
CALL METHOD container->free
    EXCEPTIONS cntl_error        = 1
           cntl_system_error = 2.
```

⚠

> Pay careful attention to the sequence in which you destroy controls at the frontend. When you destroy a container, all controls in it are automatically destroyed as well. If you have already destroyed a control and try to destroy it again, an error occurs. You can check whether a control has already been destroyed using the method is_alive [Page 485].

8. Delete the reference variables to the custom control and the control container.

```
FREE PICTURE.
FREE CONTAINER.
```

Using the Simple Tree

This section lists the functions that are specific to the simple tree.

Prerequisites

The process described here is an extension of the general process for using controls [Page 90] that is specific to the simple tree. It does not contain all of the steps required to produce a valid instance of the control.

Process Flow

➡

The program extracts are examples that do not necessarily illustrate all of the features of the control. For precise information, refer to the reference section of this documentation.

Creating the Instance

1. Define a reference variable for the simple tree:

```
DATA simple_tree TYPE REF TO cl_gui_simple_tree.
```

2. Create an instance [Page 180] of the SAP Tree:

```
CREATE OBJECT simple_tree
   EXPORTING parent  = container
         node_selection_mode = node_selection_mode
         hide_selection      = hide_selection
   EXCEPTIONS lifetime_error           = 1
          cntl_system_error        = 2
          create_error             = 3
          failed                   = 4
          illegal_node_selection_mode = 5.
```

Register the Events

3. Register the events [Page 42] for the simple tree. The control supports the following events:

Event name	Meaning
NODE_DOUBLE_CLICK	User double-clicked a node
EXPAND_NO_CHILDREN	User expanded a node that has no children
SELECTION_CHANGED	Selected node has changed
NODE_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	User selected an entry from the context menu
DEFAULT_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an empty space in the control
DEFAULT_CONTEXT_MENU_SELECT	User selected an entry from the context menu

KEYPRESS	User pressed a key. Keys for which this event is triggered have to be registered beforehand.

Using the Simple Tree

4. Insert nodes in the tree. To do this, fill a node table, then pass it to the control using the add_nodes [Page 181] method:

```
CALL METHOD simple_tree->add_nodes
   EXPORTING  table_structure_name = table_structure_name
          node_table          = node_table
   EXCEPTIONS error_in_node_table          = 1
          failed                     = 2
          dp_error                     = 3
          table_structure_name_not_found = 4.
```

5. Change existing nodes in the tree, or change the tree attributes (see Changing the Attribtues of the Control [Page 37]).

6. Query any necessary attributes of the tree and its nodes (see Finding Out the Attributes of the Control [Page 39]).

Destroying the Control

7. Destroy the custom control at the frontend. If you no longer need the control container, release it as well:

```
CALL METHOD simple_tree->free.
```

If you are working with the lifetime management [Ext.], you do not need to worry about destroying the control at the frontend yourself. It is done automatically by the system instead.

8. Delete the reference variables to the simple tree and the control container.

```
FREE simple_tree.
```

Changing the Attributes of the Control

This section lists all of the methods you can use to change the simple tree.

Inserting and Changing Nodes

Method	Description
add_nodes [Page 181]	Inserts a list of nodes
node_set_text [Page 184]	Changes the text of a node
update_nodes [Page 185]	Change the attributes of a list of nodes

Expanding Nodes

Method	Description
expand_node [Page 128]	Expands a particular node
expand_nodes [Page 129]	Expands a set of nodes
expand_root_nodes [Page 130]	Expands all root nodes

Selecting Nodes

Method	Description
set_selected_node [Page 152]	Selects a particular node
select_nodes [Page 147]	Selects a list of nodes
unselect_all [Page 154]	Deselects all selected nodes
unselect_nodes [Page 155]	Deselects a set of nodes

Deleting Nodes

Method	Description
delete_all_nodes [Page 124]	Deletes all nodes from the tree
delete_node [Page 125]	Deletes a particular node from the tree
delete_nodes [Page 126]	Deletes a set of nodes from the tree

Changing the Attributes of a Node

Method	Description
node_set_disabled [Page 136]	Deactivates nodes
node_set_expander [Page 138]	Sets the **expander** attribute.
node_set_exp_image [Page 139]	Sets expanded node icon
node_set_hidden [Page 140]	Hides a node
node_set_is_folder [Page 141]	Sets the **is_folder** attribute

Changing the Attributes of the Control

node_set_no_branch [Page 142]	Suppresses the hierarchy line to the node
node_set_n_image [Page 143]	Sets the non-expanded node icon
node_set_style [Page 144]	Sets the style of the node
node_set_dragdropid [Page 137]	Sets the drag and drop behavior of a node

Configuring Keyboard Events

Method	Description
add_key_stroke [Page 120]	Sets a key to trigger an event
remove_all_key_strokes [Page 145]	Deregisters all keys that were registered to trigger an event

Other Methods

Method	Description
ensure_visible [Page 127]	Ensures that a particular node is visible
move_node [Page 135]	Moves a node
scroll [Page 146]	Scrolls in the tree
set_ctx_menu_select_event_appl [Page 156]	Sets whether the event triggered when the user chooses an entry from a context menu should be an application event or a system event
set_has_3d_frame [Page 150]	Sets the 3D frame
set_screen_update [Page 151]	Controls the visibility of changes
set_top_node [Page 153]	Defines the topmost visible node
set_default_drop [Page 148]	Sets the drag and drop behavior for dropping on the background of the SAP Tree
set_folder_show_exp_image [Page 149]	Sets the open folder symbol

Finding Out the Attributes of the Control

This section lists all of the methods you can use to retrieve the attributes of the simple tree.

Methods for Retrieving Control Attributes

Method	Meaning
get_expanded_nodes [Page 131]	Lists all expanded nodes
get_selected_node [Page 132]	Returns the name of the selected node
get_selected_nodes [Page 133]	Lists all selected nodes
get_top_node [Page 134]	Returns the name of the topmost visible node

Registering and Processing Events

Purpose

The event mechanism of the Control Framework allows you to use handler methods in your programs to react to events triggered by the control (for example, a double-click).

Prerequisites

The following description has been generalized to apply to all custom controls. For more information specific to a particular control, refer to that control's documentation.

Process Flow

1. Assume you are working with a custom control that has the ABAP wrapper **`cl_gui_xyz`**.

```
DATA my_control TYPE REF TO cl_gui_xyz.
```

Registering Events with the Control Framework

2. Define an internal table (type **`cntl_simple_events`**) and a corresponding work area (type **`cntl_simple_event`**).

```
DATA events TYPE cntl_simple_events.
DATA wa_events TYPE  cntl_simple_event.
```

3. Now fill the event table with the relevant events. To do this, you need the event ID (**`event_id`** field). You can find this information in the Class Browser by looking at the attributes of the class **`cl_gui_xyz`**. You must also decide whether the event is to be a system event (**`appl_event = ' '`**) or an application event (**`appl_event = 'X'`**).

```
wa_events-eventid = event_id.
wa_events-appl_event = appl_event.
APPEND wa_events TO events.
```

4. You must now send the event table to the frontend so that it knows which events it has to direct to the backend.

```
CALL METHOD my_control->set_registered_events
        events = events.
```

To react to the events of you custom control, you must now specify a handler method for it. This can be either an instance method or a static method.

Processing an Event Using an Instance Method

5. Define the (local) class definition for the event handler. To do this, specify the name of the handler method (**`Event_Handler`**). You need to look at the class for the custom control cl_gui_xyz in the Class Browser to find out the name of the event (**`event_name`**) and its parameters (**`event_parameter`**). There is also a default event parameter **`sender`**, which is passed by all events. This contains the reference to the control that triggered the event.

```
CLASS lcl_event_receiver DEFINITION.
PUBLIC SECTION.
METHODS Event_Handler
  FOR EVENT event_name OF cl_gui_xyz
  IMPORTING event_parameter
         sender.
ENDCLASS.
```

6. Register the handler methods with the ABAP Objects Control Framework for the events.

```
DATA event_receiver TYPE REF TO lcl_event_receiver.
CREATE OBJECT event_receiver.
SET HANDLER event_receiver->Event_Handler
        FOR my_control.
```

Processing an Event Using a Static Method

7. Define the (local) class definition for the event handler. To do this, specify the name of the handler method (**`Event_Handler`**). You need to look at the class for the custom control cl_gui_xyz in the Class Browser to find out the name of the event (**`event_name`**) and its parameters (**`event_parameter`**).

```
CLASS lcl_event_receiver DEFINITION.
PUBLIC SECTION.
CLASS-METHODS Event_Handler
      FOR EVENT event_name OF cl_gui_xyz
      IMPORTING event_parameter
             sender.
ENDCLASS.
```

8. Register the handler methods with the ABAP Objects Control Framework for the events.

```
SET HANDLER lcl_event_receiver=>Event_Handler
        FOR my_control.
```

Processing Control Events

9. You define how you want the system to react to an event in the implementation of the handler method.

```
CLASS lcl_event_receiver IMPLEMENTATION.
METHOD Event_Handler.
* Event processing
ENDMETHOD
ENDCLASS.
```

10. If you registered your event as an application event, you need to process it using the method **`CL_GUI_CFW=>DISPATCH`**. For further information, refer to Event Handling [Ext.].

Events of the Simple Tree

Use

Certain user actions within the simple tree trigger events:

Event	Event ID CL_GUI_SIMPLE_TREE=>	Description
NODE_DOUBLE_CLICK	EVENTID_NODE_DOUBLE_CLICK	Double-click on a node
EXPAND_NO_CHILDREN	EVENTID_EXPAND_NO_CHILDREN	User expanded a node that has no children The **EXPANDER** attribute of the node must be set to **'X'**.
SELECTION_CHANGED	EVENTID_SELECTION_CHANGED	You can only use this event if you specified single selection for the tree control when you created it. Selected node has changed **Important: If you use this event, you cannot use the NODE_DOUBLE_CLICK event.**
NODE_KEYPRESS	EVENTID_NODE_KEYPRESS	The user pressed a key while a node was selected
NODE_CONTEXT_MENU_REQUEST	EVENTID_NODE_CONTEXT_MENU_REQ	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **NODE_CONTEXT_MENU_REQUEST**.	User selected an entry in the context menu for a node
DEFAULT_CONTEXT_MENU_REQUEST	EVENTID_DEF_CONTEXT_MENU_REQ	User requested a context menu with the cursor positioned on the tree background
DEFAULT_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **DEFAULT_CONTEXT_MENU_REQUEST**.	User selected an entry from the context menu for the tree background
ON_DROP_GET_FLAVOR	See Drag and Drop Events in the SAP Tree [Page 111]	There are several different drag and drop flavors

ON_DRAG	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the source object (single selection)
ON_DRAG_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the source object (multiple selection)
ON_DROP	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the context in the target object
ON_DROP_COMPLETE	See Drag and Drop Events in the SAP Tree [Page 111]	Last event before completion of the drag and drop (single selection)
ON_DROP_COMPLETE_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	Last event before completion of the drag and drop (multiple selection)

Some events also export parameters:

Event	**Parameters**	**Description**
NODE_DOUBLE_CLICK	NODE_KEY	Node on which the user double-clicked
EXPAND_NO_CHILDREN	NODE_KEY	Node without child nodes that the user tried to expand
SELECTION_CHANGED	NODE_KEY	New selected node
NODE_KEYPRESS	NODE_KEY	Node selected when the user pressed the key
	KEY	Key pressed
NODE_CONTEXT_MENU_REQUEST	NODE_KEY	Node selected when the user requested the context menu
	MENU	Menu to be displayed (must be filled in the event handler)
NODE_CONTEXT_MENU_SELECT	NODE_KEY	Node selected when the user chose an entry from the context menu
	FCODE	Function code of the selected entry in the context menu
DEFAULT_CONTEXT_MENU_REQUEST	MENU	Menu to be displayed (must be filled in the event handler)
DEFAULT_CONTEXT_MENU_SELECT	FCODE	Function code of the selected entry in the context menu

ON_DROP_GET_FLAVOR	See Drag and Drop Events in the SAP Tree [Page 111]
ON_DRAG	See Drag and Drop Events in the SAP Tree [Page 111]
ON_DRAG_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]
ON_DROP	See Drag and Drop Events in the SAP Tree [Page 111]
ON_DROP_COMPLETE	See Drag and Drop Events in the SAP Tree [Page 111]
ON_DROP_COMPLETE_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]

If you want to use events that rely on the user pressing a key (for example, **`NODE_KEYPRESS`**), you must register the keystroke using the method add_key_stroke [Page 120]. You can deregister the registered key strokes using the method remove_all_key_strokes [Page 145].

Integration

To react to an event in your ABAP program, you must have registered it. To do this, use the method set_registered_events [Page 483]. Events that are triggered but for which you are not registered are filtered by the presentation server, and not passed to the application server. **See** event handling [Ext.].

Activities

Read the general process [Page 99] for working with events in the Control Framework.

Drag and Drop

Use

Drag and drop allows the user to select an object from one part of a custom control (source) and drop it on another part of a custom control (target). An action occurs in the second part that depends on the object type. Source and target may be either the same control or different controls.

Prerequisites

For a control to support drag and drop, the control wrapper must provide drag and drop events. You must then write handler methods for these events in your program. The events are registered automatically by the relevant control wrapper.

Features

A particular drag and drop behavior is set for each custom control. This behavior may be set globally for all elements of the control (for example, SAP Textedit), or you may be able to define a different behavior for each component (for example SAP Tree). Each behavior consists of one or more descriptions.

A description has the following attributes:

- **DragSrc**: Object is the source of a drag and drop procedure
- **DropTarget**: Object is the target of a drag and drop procedure
- **Flavor**: The flavor describes the type of a drag and drop description. In a drag and drop operation, you can only drop an object onto another if both have at least one common description.
- **Effect**: Specifies whether the drag and drop operations copies or moves the object.
- **Effect_In_Ctrl**: The drop effect used when you copy or move data within the same control.

As soon as a drag event is triggered, you must use the corresponding handler method to find out the affected object.

You must also define the action that is to be carried out on the drop event. The action usually depends on the object that you drop in the control.

If you assign more than one flavor to an object, you must define which flavor is to be used. You do this in the handler for another event.

Once the drop event is finished, you can use a further event to implement additional actions. This is particularly useful for deleting the dropped object from the source after a move operation.

Activities

Whenever you provide a drag and drop function to move objects, you should always provide an *Undo* function as well. You must implement this yourself in the application.

Process Flow of a Drag and Drop Operation

Prerequisites

The following section explains how a drag and drop operation works, examining into the roles of the application server and frontend, and going on to identify the individual steps required to program drag and drop in an application.

Process Flow

Application Server

1. You create the custom control [Page 90].
2. You register the drag and drop events [Page 109].
3. You define the drag and drop behavior for the individual custom controls or their components. To do this, you create an instance [Page 494] of the class CL_DRAGDROP [Page 493]. You then assign one or more flavors [Page 495] to this instance. These describe the drag and drop behavior of the relevant custom control. During the program, you can change [Page 502], delete [Page 504], and query [Page 499] the flavors in your program. You can also initialize [Page 497] or destroy [Page 498] the entire instance.
4. You assign flavors to the custom control using specific methods of the relevant control. For further information, refer to the corresponding control documentation.

Frontend

The following steps are performed by the system at the frontend. They are only listed here so that you can understand what happens during a drag and drop operation.

5. Once the use has selected an object with the left mouse button, the drag and drop service starts.
6. The drag and drop service checks whether a drag and drop behavior has been defined for the object, and whether the object can be dragged (DragSource attribute).
7. If, according to the DragSource attribute, the object can be dragged, the drag and drop operation starts. The mouse pointer then changes automatically.
8. As long as the left mouse button remains pressed, the system continually checks whether the mouse pointer is positioned over an object in a custom control that can receive a dropped object (DropTarget attribute), and whether the flavor of that object is the same as the flavor of the source. If this is the case, the mouse pointer changes again to inform the user.
9. If the user now drops the object, an event is triggered to inform the application server.

   This concludes the drag and drop operation for the frontend. However, there has not yet been any change to the contents of the custom control.

Application Server

10. The drag and drop service of the application server creates an instance of the class CL_DRAGDROPOBJECT [Page 505]. You can use this instance (for example,

`drag_drop_object`) in all events of the drag and drop process as an event parameter. You can use it to find out the context between the events.

11. The drag and drop service checks whether the drag object and drop object have more than one flavor in common. If this is the case, the event `ONGETFLAVOR` is triggered. In the corresponding handler method, you must decide which flavor to use. You do this using the method set_flavor [Page 506].

12. Now, the drag and drop event `ONDRAG` is triggered. It has event parameters that tell you which object the user has dragged. Within the handler routine, you must pass the context (information about the source object) to the instance of the drag and drop data object created in step 9.
    `drag_drop_object->object = mydragobject.`

13. Next, the `ONDROP` event is triggered. The corresponding handler method serves to process the drag and drop data object. Here, you have to implement the changes that are to be made to the target object based on the drag and drop operation.

14. The last event of the drag and drop operation is `ONDROPCOMPLETE`. This is where you can make your last changes to the drag and drop object. In particular, you should use this event to delete the source object from the DragSource control and the corresponding data structures if you have used the drag and drop operation to move the object.

    The Example of Drag and Drop Programming [Page 114] contains an example of a drag and drop operation between a SAP Tree and a SAP Textedit.

Drag and Drop Events

This section only describes those properties of drag and drop events that apply to all controls. The individual control wrappers may augment them. You should therefore consult the relevant control documentation to see if that control has any peculiarities.

Use

There are four standard events in a drag and drop operation at which control is returned to the application program. You use the event handler methods for these events to implement the actions that should be performed during the operation.

> Some control wrappers offer additional drag and drop events. For further information, refer to the documentation of the individual controls.

Prerequisites

To be able to react to an event, you must first register it. Unlike normal event handling, you do not register drag and drop events with the Control Framework using the set_registered_events [Page 483] method Instead, they are registered automatically by the wrapper of the control that you are using.

However, you still have to specify handler methods for the events.

```
DATA tree TYPE REF TO cl_gui_simple_tree.
SET HANDLER dragdrop=>on_drag FOR tree.
```

The events are always registered as system events.

Features

In a drag and drop operation, the Control Framework does not pass any events to the application server until the object is dropped. At the application server, it is separated into up to four standard events that can occur within a drag and drop operation, as described in Process Flow of a Drag and Drop Operation [Page 107]. All events have a drag and drop data object as an event parameter. You use this parameter to manage the context of the drag and drop operation. The particular control wrapper that you are using also provides further information about the drag and drop context. For further information, refer to the documentation of the relevant control wrapper.

- **ONGETFLAVOR**: This event is only triggered if the source and target objects have more than one flavor in common. In the handler method, you must then specify which flavor should be used. To do this, use the set_flavor [Page 506] method on the drag and drop object.
  The event is triggered by the target object of the drag and drop operation.

- **ONDRAG**: This event is triggered when the drag and drop operation is complete at the frontend. When you handle this event, you must determine the context of the target object. You then pass this context to the instance of the class **CL_DRAGDROPOBJECT** that you received as an event parameter.
  The event is triggered by the source object of the drag and drop operation.

- **ONDROP**: When you handle this event, you define what should be done to the target object. To do this, use the event parameter for the context that you filled in the **ONDRAG** event. In this event, you must remember the following:

  - Within the ONDROP event, you must make a dynamic typecast. You must catch all possible exceptions of the typecast. In the exception handling you must include handling for the case where you try to assign an invalid object. In this case, you must use the abort [Page 507] method to terminate the drag and drop processing.
  - You should select the flavor you want to use so that it is possible to assign the drag and drop object to the right TypeCast.

    The event is triggered by the target object of the drag and drop operation.

- **ONDROPCOMPLETE**: Use this event to perform any further processing necessary after the end of the drag and drop operation. For example, this would be necessary following a move operation.
  The event is triggered by the source object of the drag and drop operation.

Defining Drag and Drop Events in the SAP Tree

This section explains the special considerations that apply to drag and drop operations in the SAP Tree.

Prerequisites

To be able to react to an event, you must first register it. Unlike normal event handling, you do not register drag and drop events with the Control Framework using the set_registered_events [Page 483] method Instead, they are registered automatically by the SAP Tree control wrapper.

However, you still have to specify handler methods for the events.

The events are always registered as system events.

When you fill the node table, you must also specify which nodes are enabled for drag and drop, and the flavors that the nodes should have. You do this by assigning the relevant drag and drop behavior to the field **DRAGDROPID** (see point 3 under Drag and Drop Operations [Page 107]). This requires the following steps (see also the Drag and Drop Programming Example [Page 114]).

1. Define the drag and drop behavior:

```
DATA behaviour_left TYPE REF TO cl_dragdrop.
   CREATE OBJECT behaviour_left.
   CALL METHOD behaviour_left->add
      EXPORTING
           flavor = 'Tree_move_to_Edit'
           dragsrc = 'X'
           droptarget = ' '
           effect = cl_dragdrop=>copy.
```

2. Use the get_handle [Page 501] method to return a handle to the drag and drop behavior:

```
   CALL METHOD behaviour_left->get_handle
       IMPORTING handle = handle_tree.
```

3. Assign the handle to the **DRAGDROPID** field of the corresponding entry in the node table:

```
 node-dragdropid = handle_tree.      " handle of behaviour
```

Entries with the type **tree->item_class_checkbox** (checkboxes), **tree->item_class_button** (pushbuttons) and **tree->item_class_link** (links) cannot be the source object of a drag and drop operation.

Features

The following table contains the events used in drag and drop:

Event	Description
ON_DROP_GET_FLAVOR	See the event **ONGETFLAVOR** under Drag and Drop Events [Page 109]

ON_DRAG	See the event **ONDRAG** under Drag and Drop Events [Page 109] For trees without multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_SINGLE**).
ON_DRAG_MULTIPLE	See the event **ONDRAG** under Drag and Drop Events [Page 109] For trees with multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_MULTIPLE**).
ON_DROP	See the event **ONDROP** under Drag and Drop Events [Page 109]
ON_DROP_COMPLETE	See the event **ONDROPCOMPLETE** under Drag and Drop Events [Page 109] For trees without multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_SINGLE**).
ON_DROP_COMPLETE_MULTIPLE	See the event **ONDROPCOMPLETE** under Drag and Drop Events [Page 109] For trees with multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_MULTIPLE**).

The individual events have the following parameters:

Event	**Event parameter**	**Description**
ON_DROP_GET_FLAVOR	NODE_KEY	Technical name of the node onto which the source object was dragged
	DRAG_DROP_OBJECT	Data object describing the source object
	FLAVORS	Shared flavors of the drag and drop operation
ON_DRAG	NODE_KEY	Technical name of the node selected as the source object
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DRAG_MULTIPLE	NODE_KEY_TABLE	Table of nodes selected as source obejcts

	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP	NODE_KEY	Technical name of the node onto which the source object was dragged
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP_COMPLETE	NODE_KEY	Technical name of the node selected as the source object
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP_COMPLETE_MULTIPLE	NODE_KEY_TABLE	Table of nodes selected as source obejcts
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object

Example of Drag and Drop Programming

This example program uses a SAP Simple Tree Control and a SAP Textedit Control. The aim is to enable the user to move texts from the tree control into the textedit control.

The example has the program name **RSDEMO_DRAG_DROP_EDIT_TREE**.

```
*&-------------------------------------------------------------------*
*& Report  RSDEMO_DRAG_DROP_EDIT_TREE                              *&
*--------------------------------------------------------------------*
REPORT  rsdemo_drag_drop_edit_tree    .
DATA ok_code TYPE sy-ucomm.
DATA node_itab LIKE node_str OCCURS 0.
DATA node LIKE node_str.
DATA container TYPE REF TO cl_gui_custom_container.
DATA splitter TYPE REF TO cl_gui_easy_splitter_container.
DATA right TYPE REF TO cl_gui_container.
DATA left  TYPE REF TO cl_gui_container.
DATA editor TYPE REF TO cl_gui_textedit.
DATA tree TYPE REF TO cl_gui_simple_tree.
DATA behaviour_left TYPE REF TO cl_dragdrop.
DATA behaviour_right TYPE REF TO cl_dragdrop.
DATA handle_tree TYPE i.
*--------------------------------------------------------------------*
*       CLASS lcl_treeobject DEFINITION
*  container class for drag object
*--------------------------------------------------------------------*
CLASS lcl_drag_object DEFINITION.
  PUBLIC SECTION.
    DATA text TYPE mtreesnode-text.
ENDCLASS.
*---------------------------------------------------------------------*
*       CLASS dragdrop_receiver DEFINITION
*  event handler class for drag&drop events
*---------------------------------------------------------------------*
CLASS lcl_dragdrop_receiver DEFINITION.
  PUBLIC SECTION.
    METHODS:
       flavor_select FOR EVENT on_get_flavor OF cl_gui_textedit
                      IMPORTING index line pos flavors dragdrop_object,
       left_drag FOR EVENT on_drag OF cl_gui_simple_tree
                      IMPORTING node_key drag_drop_object,
       right_drop FOR EVENT ON_DROP OF cl_gui_textedit
                      IMPORTING index line pos dragdrop_object,
       drop_complete FOR EVENT on_drop_complete OF cl_gui_simple_tree
                      IMPORTING node_key drag_drop_object.
ENDCLASS.
START-OF-SELECTION.
  CALL SCREEN 100.
*&-------------------------------------------------------------------*
*&      Module  START  OUTPUT
*&-------------------------------------------------------------------*
MODULE start OUTPUT.
```

```
  SET PF-STATUS 'BASE'.
  IF container is initial.
    CREATE OBJECT container
        EXPORTING container_name = 'CONTAINER'.
    CREATE OBJECT splitter
        EXPORTING parent = container
                  orientation = 1.
    left = splitter->top_left_container.
    right = splitter->bottom_right_container.
    CREATE OBJECT editor
        EXPORTING parent = right.
    CREATE OBJECT tree
        EXPORTING parent = left
                  node_selection_mode = tree->node_sel_mode_single.
* Definition of drag drop behaviour for tree
    CREATE OBJECT behaviour_left.
    CALL METHOD behaviour_left->add
        EXPORTING
              flavor = 'Tree_move_to_Edit'
              dragsrc = 'X'
              droptarget = ' '
              effect = cl_dragdrop=>copy.
   CALL METHOD behaviour_left->add
        EXPORTING
              flavor = 'Tree_copy_to_Edit'
              dragsrc = 'X'
              droptarget = ' '
              effect = cl_dragdrop=>copy.
    CALL METHOD behaviour_left->get_handle
         IMPORTING handle = handle_tree.
* Drag Drop behaviour of tree control nodes are defined in the node
* structure
    PERFORM fill_tree.
    CALL METHOD tree->add_nodes
         EXPORTING node_table = node_itab
                   table_structure_name = 'NODE_STR'.
* Definition of drag drop behaviour for tree
    CREATE OBJECT behaviour_right.
 CALL METHOD behaviour_right->add
        EXPORTING
              flavor = 'Tree_move_to_Edit'
              dragsrc = ' '
              droptarget = 'X'
              effect = cl_dragdrop=>copy.
 CALL METHOD behaviour_right->add
        EXPORTING
              flavor = 'Tree_copy_to_Edit'
              dragsrc = ' '
              droptarget = 'X'
              effect = cl_dragdrop=>copy.
    CALL METHOD editor->set_dragdrop
         EXPORTING dragdrop = behaviour_right.
```

```
* registration of drag and drop events

    SET HANDLER dragdrop=>flavor_select FOR editor.
    SET HANDLER dragdrop=>left_drag FOR tree.
    SET HANDLER dragdrop=>right_drop FOR editor.
    SET HANDLER dragdrop=>drop_complete for TREE.
  ENDIF.
ENDMODULE.                             " START  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  EXIT  INPUT
*&---------------------------------------------------------------------*
MODULE exit INPUT.
  LEAVE PROGRAM.
ENDMODULE.                             " EXIT  INPUT
*&---------------------------------------------------------------------*
*&      Form  fill_tree
*&---------------------------------------------------------------------*
FORM fill_tree.
  DATA: node LIKE mtreesnode.
  CLEAR node.
  node-node_key = 'Root'.
  node-isfolder = 'X'.
  node-text = 'Text'.
  node-dragdropid = ' '.
  APPEND node TO node_itab.
  CLEAR node.
  node-node_key = 'Child1'.
  node-relatkey = 'Root'.
  node-relatship = cl_gui_simple_tree=>relat_last_child.
  node-text = 'DragDrop Text 1'.
  node-dragdropid = handle_tree.       " handle of behaviour
  APPEND node TO node_itab.
  CLEAR node.
  node-node_key = 'Child2'.
  node-relatkey = 'Root'.
  node-relatship = cl_gui_simple_tree=>relat_last_child.
  node-text = 'DragDrop Text 2'.
  node-dragdropid = handle_tree.       " handle of behaviour
  APPEND node TO node_itab.
ENDFORM.                               " fill_tree
*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CALL METHOD cl_gui_cfw=>dispatch.
ENDMODULE.                             " USER_COMMAND_0100  INPUT
*---------------------------------------------------------------------*
*       CLASS DRAGDROP_RECEIVER IMPLEMENTATION
*---------------------------------------------------------------------*
CLASS lcl_dragdrop_receiver IMPLEMENTATION.
  METHOD flavor_select. " set the right flavor
    IF line > 5.
      SEARCH flavors FOR 'Tree_move_to_Edit'.
      IF sy-subrc = 0.
        CALL METHOD dragDROP_OBJECT->SET_FLAVOR

             EXPORTING newflavor = 'Tree_move_to_Edit'.
      ELSE.
        CALL METHOD dragdrop_object->abort.
      ENDIF.
    ELSE.
      SEARCH flavors FOR 'Tree_copy_to_Edit'.
      IF sy-subrc = 0.
        CALL METHOD dragdrop_object->set_flavor
             EXPORTING newflavor = 'Tree_copy_to_Edit'.
      ELSE.
        CALL METHOD dragdrop_object->abort.
      ENDIF.
    ENDIF.
  ENDMETHOD.
  METHOD left_drag. " define drag object
    DATA drag_object TYPE REF TO lcl_drag_object.
    READ TABLE node_itab WITH KEY node_key = node_key
                         INTO node.
    CREATE OBJECT drag_object.
    drag_object->text = node-text.
    drag_drop_object->object = drag_object.
ENDMETHOD.
  METHOD right_drop. " action in the drop object
    DATA textline(256).
    DATA text_table LIKE STANDARD TABLE OF textline.
    DATA drag_object TYPE REF TO lcl_drag_object.
    CATCH SYSTEM-EXCEPTIONS move_cast_error = 1.
      drag_object ?= dragdrop_object->object.
    ENDCATCH.
    IF sy-subrc = 1.
      " data object has unexpected class
                                   " => cancel Drag & Drop operation
      CALL METHOD dragdrop_object->abort.
      EXIT.
    ENDIF.
    CALL METHOD editor->get_text_as_stream
         IMPORTING text        = text_table.
* Synchronize Automation Queue after Get Methods
    CALL METHOD cl_gui_cfw=>flush.
    textline = drag_object->text.
* Insert text in internal table
    INSERT textline INTO text_table INDEX 1.
* Send modified table to frontend
    CALL METHOD editor->set_text_as_stream
         EXPORTING  text = text_table
         EXCEPTIONS error_dp        = 1
                    error_dp_create = 2.
  ENDMETHOD.
  METHOD drop_complete. " do something after drop
    IF drag_drop_object->flavor = 'Tree_move_to_Edit'.
```

```
      CALL METHOD tree->delete_node
           EXPORTING node_key = node_key.
      delete node_itab where node_key = node_key.

    ENDIF.
    ENDMETHOD.
ENDCLASS.
```

The Column Tree

Definition

You crate a simple tree with reference to the class **cl_gui_column_tree**:

DATA column_tree TYPE REF TO cl_gui_column_tree.

According to the inheritance hierarchy, you can now access the methods of the following classes:

- **cl_gui_object** and **cl_gui_control** (see Methods of the ABAP Objects Control Framework [Page 471]).
- **cl_tree_control_base** (see Methods of the class CL_TREE_CONTROL_BASE [Page 119]).
- **cl_item_control_base** (see Methods of the class CL_ITEM_TREE_CONTROL [Page 157]).
- **cl_gui_column_tree** (see Methods of the Class CL_GUI_COLUMN_TREE [Page 203]).

Use

The program **sapcolumn_tree_control_demo** demonstrates how to use the simple tree.

For details of the attributes of the column tree, refer to the Overview of Tree Classes [Page 18].

Creating a Control: SAP Picture Example

Prerequisites

The following process applies to all SAP custom controls. The programming examples use the SAP Picture Control. However, to apply the example to other controls, you would only have to change the name of the control class.

The example also assumes that you are using the custom control in a Custom Container. The SAP Container documentation contains details of further scenarios.

Process Flow

Create the Instance

9. Define a reference variable for the Custom Container in which you want to place the custom control (**see** SAP Container [Ext.]).

```
DATA container TYPE REF TO cl_gui_custom_container.
```

10. Define a reference variable for the SAP Picture:

```
DATA picture TYPE REF TO cl_gui_picture.
```

11. Create the Custom Container. You must already have created the area **'CUSTOM'** for the Custom Container in the Screen Painter. When you create the container, you must also specify its lifetime [Ext.] (see constructor [Ext.]).

```
CREATE OBJECT container
   EXPORTING container_name = 'CUSTOM'

          lifetime       = lifetime.
```

12. Create the SAP Picture Control. You can also specify a lifetime for the SAP Picture, but it must not be longer than that of its container.

```
CREATE OBJECT picture
    EXPORTING parent  = container

          lifetime = lifetime.
```

Register the Events

13. There are three steps: Registering the events with the Control Framework, defining a handler method, and registering the hander method. These steps are explained under Registering and Processing Events [Page 99].

Use the Control

14. These steps are control-specific and therefore not described here.

Destroy the Control

The lifetime management [Ext.] is normally responsible for destroying any controls you use. However, the following two steps allow you to destroy the control yourself:

15. Use the method free [Page 480] to destroy the Custom Control at the frontend. If you no longer need the control container, release it as well:

```
CALL METHOD picture->free
    EXCEPTIONS cntl_error       = 1
           cntl_system_error = 2.
CALL METHOD container->free
    EXCEPTIONS cntl_error       = 1
           cntl_system_error = 2.
```

⚠

Pay careful attention to the sequence in which you destroy controls at the frontend. When you destroy a container, all controls in it are automatically destroyed as well. If you have already destroyed a control and try to destroy it again, an error occurs. You can check whether a control has already been destroyed using the method is_alive [Page 485].

16. Delete the reference variables to the custom control and the control container.

```
FREE PICTURE.
FREE CONTAINER.
```

Using the Column Tree

This section lists the functions that are specific to the column tree.

Prerequisites

The process described here is an extension of the general process for using controls [Page 90] that is specific to the simple tree. It does not contain all of the steps required to produce a valid instance of the control.

Process Flow

The program extracts are examples that do not necessarily illustrate all of the features of the control. For precise information, refer to the reference section of this documentation.

Creating the Instance

9. Define a reference variable for the column tree:

```
DATA column_tree TYPE REF TO cl_gui_column_tree.
```

10. Define a work area for the hierarchy heading by referring to the structure **treev_hhdr**.

```
DATA hierarchy_header TYPE treev_hhdr.
```

11. Fill the work area for the hierarchy heading. You can set the width (**width** and **width_pix**), the text (**heading**), an icon (**t_image**) and a tool tip (**tooltip**). There are also methods that allow you to change these attributes later on.

```
hierarchy_header-heading = 'Title'.
hierarchy_header-width = 30.
```

12. Create an instance [Page 204] of the SAP Tree:

```
CREATE OBJECT column_tree
    EXPORTING parent  = container
            node_selection_mode   = node_selection_mode
            hide_selection        = hide_selection
            item_selection        = item_selection
            hierarchy_column_name = hierarchy_column_name
            hierarchy_header      = hierarchy_header
    EXCEPTIONS lifetime_error           =  1
             cntl_system_error          = 2
             create_error             = 3
             illegal_node_selection_mode = 4
             failed                   = 5
             illegal_column_name        = 6.
```

Register the Events

13. Register the events [Page 101] for the column tree. The control supports the following events:

Event name	Description
NODE_DOUBLE_CLICK	User double-clicked a node
EXPAND_NO_CHILDREN	User expanded a node that has no children
SELECTION_CHANGED	Selected node has changed
NODE_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	User selected an entry from the context menu
DEFAULT_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an empty space in the control
DEFAULT_CONTEXT_MENU_SELECT	User selected an entry from the context menu
HEADER_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on the heading
HEADER_CONTEXT_MENU_SELECT	User selected an entry from the context menu
ITEM_KEYPRESS	User pressed a key while an entry was selected.
NODE_KEYPRESS	User pressed a key while an entry was selected.
HEADER_CLICK	User clicked a heading

If you set the parameter `item_selection = 'X'` when you created the instance, you can also react to the following events:

Event name	Description
BUTTON_CLICK	The user clicked an item with the class BUTTON
LINK_CLICK	The user clicked an item with the class LINK
CHECKBOX_CHANGE	The user clicked an item with the class CHECKBOX
ITEM_DOUBLE_CLICK	The user double-clicked an item
ITEM_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an item
ITEM_CONTEXT_MENU_SELECT	User selected an entry from the context menu for an item

Using the Column Tree

14. Insert nodes in the tree. To do this, first fill a node table and an item table, then pass them to the control using the add_nodes_and_items [Page 158] method.

```
CALL METHOD column_tree->add_nodes_and_items
   EXPORTING  node_table              = node_table
          item_table            = item_table
          item_table_structure_name = item_table_structure_name
   EXCEPTIONS failed                   = 1
          cntl_system_error           = 2
          error_in_tables            = 3
```

```
dp_error                  = 4
table_structure_name_not_found = 5.
```

15. Change existing nodes in the tree, or change the tree attributes (see Changing the Attribtues of the Control [Page 64]).

16. Query any necessary attributes of the tree and its nodes (see Finding Out the Attributes of the Control [Page 68]).

Destroying the Control

17. Destroy the custom control at the frontend. If you no longer need the control container, release it as well:

```
CALL METHOD column_tree->free.
```

If you are working with the lifetime management [Ext.], you do not need to worry about destroying the control at the frontend yourself. It is done automatically by the system instead.

18. Delete the reference variables to the simple tree and the control container.

```
FREE column_tree.
```

Changing the Attributes of the Control

This section lists all of the methods you can use to change the column tree.

Inserting, Changing, and Deleting Items (With Nodes)

Method	Description
add_nodes_and_items [Page 158]	Adds a set of items (and their nodes)
delete_all_items_of_nodes [Page 162]	Deletes all items for a list of nodes
delete_items [Page 163]	Deletes a set of items
update_nodes_and_items [Page 175]	Changes a list of entries (and their nodes)

Changing Individual Items

Method	Description
item_set_chosen [Page 165]	Sets a checkbox in the tree to selected
item_set_disabled [Page 166]	Deactivates an entry in the table
item_set_editable [Page 167]	Sets whether a checkbox can be changed
item_set_font [Page 168]	Sets the font for the item
item_set_hidden [Page 169]	Makes an item invisible
item_set_style [Page 170]	Sets the style of an item
item_set_text [Page 171]	Changes the text of an item
item_set_t_image [Page 172]	Changes the icon of an item

Selecting a Single Item

Method	Description
select_item [Page 173]	Selects a single item

Expanding Nodes

Method	Description
expand_node [Page 128]	Expands a particular node
expand_nodes [Page 129]	Expands a set of nodes
expand_root_nodes [Page 130]	Expands all root nodes

Selecting Nodes

Method	Description
set_selected_node [Page 152]	Selects a particular node
select_nodes [Page 147]	Selects a list of nodes
unselect_all [Page 154]	Deselects all seleceted nodes and items

unselect_nodes [Page 155]	Deselects a set of nodes

Deleting Nodes

Method	Description
delete_all_nodes [Page 124]	Deletes all nodes from the tree
delete_node [Page 125]	Deletes a particular node from the tree
delete_nodes [Page 126]	Deletes a set of nodes from the tree

Changing the Attributes of a Node

Method	Description
node_set_disabled [Page 136]	Deactivates nodes
node_set_expander [Page 138]	Sets the **`expander`** attribute.
node_set_exp_image [Page 139]	Sets expanded node icon
node_set_hidden [Page 140]	Hides a node
node_set_is_folder [Page 141]	Sets the **`is_folder`** attribute
node_set_no_branch [Page 142]	Sets whether the hierarchy line is drawn to the node
node_set_n_image [Page 143]	Sets the non-expanded node icon
node_set_style [Page 144]	Sets the style of the node
node_set_dragdropid [Page 137]	Sets the drag and drop behavior of a node

Inserting, Deleting and Changing Columns

Method	Description
add_column [Page 206]	Adds a column
add_hierarchy_column [Page 208]	Add a column below the hierarchy heading
delete_column [Page 217]	Delete column
insert_column [Page 225]	Inserts a column at a particular position
insert_hierarchy_column [Page 227]	Inserts a column at a particular position below the hierarchy header

Changing Column Attributes

Method	Description
column_set_disabled [Page 211]	Deactivates a column
column_set_heading_image [Page 212]	Changes the icon of the heading
column_set_heading_text [Page 213]	Changes the text of the heading
column_set_heading_tooltip [Page 214]	Changes the tooltip of the heading

Changing the Attributes of the Control

column_set_hidden [Page 215]	Hides a column
column_set_width [Page 216]	Changes the width of the column
adjust_column_width [Page 209]	Adjusts the width of a column
update_column [Page 229]	Changes a set of attributes of a column

Changing the Attributes of the Hierarchy Heading

Method	Description
hierarchy_header_set_t_image [Page 223]	Changes the icon of the hierarchy heading
hierarchy_header_set_text [Page 221]	Changes the text of the hieararchy heading
hierarchy_header_set_tooltip [Page 222]	Changes the tooltip of the hierarchy heading
hierarchy_header_set_width [Page 224]	Changes the width of the hierarchy heading
hierarchy_header_adjust_width [Page 219]	Adjusts the width of the hierarchy heading

Setting the Sequence of Columns

Method	Description
set_column_order [Page 228]	Setting the Sequence of Columns

Configuring Keyboard Events

Method	Description
add_key_stroke [Page 120]	Sets a key to trigger an event
remove_all_key_strokes [Page 145]	Deregisters all keys that were registered to trigger an event

Other Methods

Method	Description
ensure_visible [Page 127]	Ensures that a particular node is visible
move_node [Page 135]	Moves a node
scroll [Page 146]	Scrolls in the tree
set_ctx_menu_select_event_appl [Page 156]	Sets whether the event triggered when the user chooses an entry from a context menu should be an application event or a system event
set_has_3d_frame [Page 150]	Sets the 3D frame
set_screen_update [Page 151]	Controls the visibility of changes
set_top_node [Page 153]	Defines the topmost visible node
set_min_node_height [Page 174]	Sets the minimum height of a node

set_default_drop [Page 148]	Sets the drag and drop behavior for dropping on the background of the SAP Tree
set_folder_show_exp_image [Page 149]	Sets the open folder symbol

Finding Out the Attributes of the Control

This section lists all of the methods you can use to retrieve the attributes of the column tree.

Methods for Retrieving Control Attributes

Method	Description
get_expanded_nodes [Page 131]	Lists all expanded nodes
get_selected_node [Page 132]	Returns the name of the selected node
get_selected_nodes [Page 133]	Lists all selected nodes
get_top_node [Page 134]	Name of the topmost visible node
get_selected_item [Page 164]	Name of the selected item
hierarchy_header_get_width [Page 220]	Width of the hierarchy heading
column_get_width [Page 210]	Width of a column
get_column_order [Page 218]	Sequence of the columns

Registering and Processing Events

Purpose

The event mechanism of the Control Framework allows you to use handler methods in your programs to react to events triggered by the control (for example, a double-click).

Prerequisites

The following description has been generalized to apply to all custom controls. For more information specific to a particular control, refer to that control's documentation.

Process Flow

11. Assume you are working with a custom control that has the ABAP wrapper **`cl_gui_xyz`**.

```
DATA my_control TYPE REF TO cl_gui_xyz.
```

Registering Events with the Control Framework

12. Define an internal table (type **`cntl_simple_events`**) and a corresponding work area (type **`cntl_simple_event`**).

```
DATA events TYPE cntl_simple_events.
DATA wa_events TYPE  cntl_simple_event.
```

13. Now fill the event table with the relevant events. To do this, you need the event ID (**`event_id`** field). You can find this information in the Class Browser by looking at the attributes of the class **`cl_gui_xyz`**. You must also decide whether the event is to be a system event (**`appl_event = ' '`**) or an application event (**`appl_event = 'X'`**).

```
wa_events-eventid = event_id.
wa_events-appl_event = appl_event.
APPEND wa_events TO events.
```

14. You must now send the event table to the frontend so that it knows which events it has to direct to the backend.

```
CALL METHOD my_control->set_registered_events
        events = events.
```

To react to the events of you custom control, you must now specify a handler method for it. This can be either an instance method or a static method.

Processing an Event Using an Instance Method

15. Define the (local) class definition for the event handler. To do this, specify the name of the handler method (**`Event_Handler`**). You need to look at the class for the custom control cl_gui_xyz in the Class Browser to find out the name of the event (**`event_name`**) and its parameters (**`event_parameter`**). There is also a default event parameter **`sender`**, which is passed by all events. This contains the reference to the control that triggered the event.

```
CLASS lcl_event_receiver DEFINITION.
PUBLIC SECTION.
METHODS Event_Handler
  FOR EVENT event_name OF cl_gui_xyz
  IMPORTING event_parameter
         sender.
ENDCLASS.
```

16. Register the handler methods with the ABAP Objects Control Framework for the events.

```
DATA event_receiver TYPE REF TO lcl_event_receiver.
CREATE OBJECT event_receiver.
SET HANDLER event_receiver->Event_Handler
        FOR my_control.
```

Processing an Event Using a Static Method

17. Define the (local) class definition for the event handler. To do this, specify the name of the handler method (**Event_Handler**). You need to look at the class for the custom control cl_gui_xyz in the Class Browser to find out the name of the event (**event_name**) and its parameters (**event_parameter**).

```
CLASS lcl_event_receiver DEFINITION.
PUBLIC SECTION.
CLASS-METHODS Event_Handler
      FOR EVENT event_name OF cl_gui_xyz
      IMPORTING event_parameter
             sender.
ENDCLASS.
```

18. Register the handler methods with the ABAP Objects Control Framework for the events.

```
SET HANDLER lcl_event_receiver=>Event_Handler
        FOR my_control.
```

Processing Control Events

19. You define how you want the system to react to an event in the implementation of the handler method.

```
CLASS lcl_event_receiver IMPLEMENTATION.
METHOD Event_Handler.
* Event processing
ENDMETHOD
ENDCLASS.
```

20. If you registered your event as an application event, you need to process it using the method **CL_GUI_CFW=>DISPATCH**. For further information, refer to Event Handling [Ext.].

Events of the Column Tree and List Tree

Use

Certain user actions on the column tree and list tree trigger events:

Event	Event ID CL_ITEM_TREE_CONTROL=>	Description
NODE_DOUBLE_CLICK	EVENTID_NODE_DOUBLE_CLICK	Double-click on a node
NODE_KEYPRESS	EVENTID_NODE_KEYPRESS	The user pressed a key while a node was selected
EXPAND_NO_CHILDREN	EVENTID_EXPAND_NO_CHILDREN	User expanded a node that has no children
SELECTION_CHANGED	EVENTID_SELECTION_CHANGED	You can only use this event if you specified single node selection and `ITEM_SELECTION = ' '` when you created the control. Selected node has changed **Important: If you use this event, you cannot use the NODE_DOUBLE_CLICK event.**
NODE_CONTEXT_MENU_REQUEST	EVENTID_NODE_CONTEXT_MENU_REQ	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **`NODE_CONTEXT_MENU_REQUEST`**.	User selected an entry in the context menu for a node
DEFAULT_CONTEXT_MENU_REQUEST	EVENTID_DEF_CONTEXT_MENU_REQ	User requested a context menu with the cursor positioned on the tree background
DEFAULT_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **`DEFAULT_CONTEXT_MENU_REQUEST`**.	User selected an entry from the context menu for the tree background
HEADER_CONTEXT_MENU_REQUEST	EVENTID_HEADER_CONTEXT_MEN_REQ	User requested a context menu with the cursor positioned on a heading

HEADER_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **EVENTID_HEADER_CONTEXT_MEN_REQ.**	User selected an entry from the context menu for the tree background
HEADER_CLICK	EVENTID_HEADER_CLICK	User clicked a heading
ON_DROP_GET_FLAVOR	See Drag and Drop Events in the SAP Tree [Page 111]	There are several different drag and drop flavors
ON_DRAG	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the source object (single selection)
ON_DRAG_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the source object (multiple selection)
ON_DROP	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the context in the target object
ON_DROP_COMPLETE	See Drag and Drop Events in the SAP Tree [Page 111]	Last event before completion of the drag and drop (single selection)
ON_DROP_COMPLETE_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	Last event before completion of the drag and drop (multiple selection)

If you set the parameter **item_selection = 'X'** when you created the instance, you can also react to the following events:

Event	**Event ID** CL_ITEM_TREE_CONTROL=>	**Description**
ITEM_DOUBLE_CLICK	EVENTID_ITEM_DOUBLE_CLICK	The user double-clicked an item
ITEM_KEYPRESS	EVENTID_ITEM_KEYPRESS	The user pressed a key while a node was selected
BUTTON_CLICK	EVENTID_BUTTON_CLICK	The user clicked an item with type BUTTON
LINK_CLICK	EVENTID_LINK_CLICK	The user clicked an item with type LINK
CHECKBOX_CHANGE	EVENTID_CHECKBOX_CHANGE	The user clicked an item with type CHECKBOX
ITEM_CONTEXT_MENU_REQUEST	EVENTID_ITEM_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on a node
ITEM_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **ITEM_CONTEXT_MENU_REQUEST.**	User selected an entry from the context menu

Some events also export parameters:

Event	Parameters	Description
NODE_DOUBLE_CLICK	NODE_KEY	Node on which the user double-clicked
NODE_KEYPRESS	NODE_KEY	Node selected when the user pressed the key
	KEY	Key pressed
EXPAND_NO_CHILDREN	NODE_KEY	Node without child nodes that the user tried to expand
SELECTION_CHANGED	NODE_KEY	New selected node
NODE_CONTEXT_MENU_REQUEST	NODE_KEY	Node selected when the user requested the context menu
	MENU	Menu to be displayed (must be filled in the event handler)
NODE_CONTEXT_MENU_SELECT	NODE_KEY	Node selected when the user chose an entry from the context menu
	FCODE	Function code of the selected entry in the context menu
HEADER_CLICK	HEADER_NAME	Name of the heading clicked by the user
HEADER_CONTEXT_MENU_REQUEST	HEADER_NAME	Heading selected when the user requested the context menu
	MENU	Menu to be displayed (must be filled in the event handler)
HEADER_CONTEXT_MENU_SELECT	HEADER_NAME	Heading selected when the user selected from the context menu
	FCODE	Function code of the selected entry in the context menu
CHECKBOX_CHANGE	NODE_KEY	Name of the node
	ITEM_NAME	Name of the item clicked by the user
	CHECKED	'X': **Checkbox selected** ' ': **Checkbox not selected**
ITEM_DOUBLE_CLICK	NODE_KEY	Name of the node

	ITEM_NAME	Name of the item clicked by the user
ITEM_CONTEXT_MENU_REQUEST	NODE_KEY	Name of the node
	ITEM_NAME	Name of the item on which the context menu was request
	MENU	Menu to be displayed (must be filled in the event handler)
ITEM_CONTEXT_MENU_SELECT	NODE_KEY	Name of the node
	ITEM_NAME	Name of the item on which the context menu was request
	FCODE	Function code of the selected entry in the context menu
ITEM_KEYPRESS	NODE_KEY	Name of the node
	ITEM_NAME	Name of the item selected when the user pressed a key
	KEY	Key pressed
DEFAULT_CONTEXT_MENU_REQUEST	MENU	Menu to be displayed (must be filled in the event handler)
DEFAULT_CONTEXT_MENU_SELECT	FCODE	Function code of the selected entry in the context menu
ON_DROP_GET_FLAVOR	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DRAG	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DRAG_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DROP	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DROP_COMPLETE	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DROP_COMPLETE_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	

If you want to use events that rely on the user pressing a key (for example, **`NODE_KEYPRESS`**), you must register the keystroke using the method add_key_stroke [Page 120]. You can deregister the registered key strokes using the method remove_all_key_strokes [Page 145].

Integration

To react to an event in your ABAP program, you must have registered it. To do this, use the method set_registered_events [Page 483]. Events that are triggered but for which you are not registered are filtered by the presentation server, and not passed to the application server. **See** event handling [Ext.].

Activities

Read the general process [Page 90] for working with events in the Control Framework.

Drag and Drop

Use

Drag and drop allows the user to select an object from one part of a custom control (source) and drop it on another part of a custom control (target). An action occurs in the second part that depends on the object type. Source and target may be either the same control or different controls.

Prerequisites

For a control to support drag and drop, the control wrapper must provide drag and drop events. You must then write handler methods for these events in your program. The events are registered automatically by the relevant control wrapper.

Features

A particular drag and drop behavior is set for each custom control. This behavior may be set globally for all elements of the control (for example, SAP Textedit), or you may be able to define a different behavior for each component (for example SAP Tree). Each behavior consists of one or more descriptions.

A description has the following attributes:

- **DragSrc**: Object is the source of a drag and drop procedure
- **DropTarget**: Object is the target of a drag and drop procedure
- **Flavor**: The flavor describes the type of a drag and drop description. In a drag and drop operation, you can only drop an object onto another if both have at least one common description.
- **Effect**: Specifies whether the drag and drop operations copies or moves the object.
- **Effect_In_Ctrl**: The drop effect used when you copy or move data within the same control.

As soon as a drag event is triggered, you must use the corresponding handler method to find out the affected object.

You must also define the action that is to be carried out on the drop event. The action usually depends on the object that you drop in the control.

If you assign more than one flavor to an object, you must define which flavor is to be used. You do this in the handler for another event.

Once the drop event is finished, you can use a further event to implement additional actions. This is particularly useful for deleting the dropped object from the source after a move operation.

Activities

Whenever you provide a drag and drop function to move objects, you should always provide an *Undo* function as well. You must implement this yourself in the application.

Process Flow of a Drag and Drop Operation

Prerequisites

The following section explains how a drag and drop operation works, examining into the roles of the application server and frontend, and going on to identify the individual steps required to program drag and drop in an application.

Process Flow

Application Server

15. You create the custom control [Page 90].

16. You register the drag and drop events [Page 109].

17. You define the drag and drop behavior for the individual custom controls or their components. To do this, you create an instance [Page 494] of the class CL_DRAGDROP [Page 493]. You then assign one or more flavors [Page 495] to this instance. These describe the drag and drop behavior of the relevant custom control. During the program, you can change [Page 502], delete [Page 504], and query [Page 499] the flavors in your program. You can also initialize [Page 497] or destroy [Page 498] the entire instance.

18. You assign flavors to the custom control using specific methods of the relevant control. For further information, refer to the corresponding control documentation.

Frontend

The following steps are performed by the system at the frontend. They are only listed here so that you can understand what happens during a drag and drop operation.

19. Once the use has selected an object with the left mouse button, the drag and drop service starts.

20. The drag and drop service checks whether a drag and drop behavior has been defined for the object, and whether the object can be dragged (DragSource attribute).

21. If, according to the DragSource attribute, the object can be dragged, the drag and drop operation starts. The mouse pointer then changes automatically.

22. As long as the left mouse button remains pressed, the system continually checks whether the mouse pointer is positioned over an object in a custom control that can receive a dropped object (DropTarget attribute), and whether the flavor of that object is the same as the flavor of the source. If this is the case, the mouse pointer changes again to inform the user.

23. If the user now drops the object, an event is triggered to inform the application server.

This concludes the drag and drop operation for the frontend. However, there has not yet been any change to the contents of the custom control.

Application Server

24. The drag and drop service of the application server creates an instance of the class CL_DRAGDROPOBJECT [Page 505]. You can use this instance (for example,

drag_drop_object) in all events of the drag and drop process as an event parameter. You can use it to find out the context between the events.

25. The drag and drop service checks whether the drag object and drop object have more than one flavor in common. If this is the case, the event **ONGETFLAVOR** is triggered. In the corresponding handler method, you must decide which flavor to use. You do this using the method set_flavor [Page 506].

26. Now, the drag and drop event **ONDRAG** is triggered. It has event parameters that tell you which object the user has dragged. Within the handler routine, you must pass the context (information about the source object) to the instance of the drag and drop data object created in step 9.
    ```
    drag_drop_object->object = mydragobject.
    ```

27. Next, the **ONDROP** event is triggered. The corresponding handler method serves to process the drag and drop data object. Here, you have to implement the changes that are to be made to the target object based on the drag and drop operation.

28. The last event of the drag and drop operation is **ONDROPCOMPLETE**. This is where you can make your last changes to the drag and drop object. In particular, you should use this event to delete the source object from the DragSource control and the corresponding data structures if you have used the drag and drop operation to move the object.

    The Example of Drag and Drop Programming [Page 114] contains an example of a drag and drop operation between a SAP Tree and a SAP Textedit.

Drag and Drop Events

This section only describes those properties of drag and drop events that apply to all controls. The individual control wrappers may augment them. You should therefore consult the relevant control documentation to see if that control has any peculiarities.

Use

There are four standard events in a drag and drop operation at which control is returned to the application program. You use the event handler methods for these events to implement the actions that should be performed during the operation.

> Some control wrappers offer additional drag and drop events. For further information, refer to the documentation of the individual controls.

Prerequisites

To be able to react to an event, you must first register it. Unlike normal event handling, you do not register drag and drop events with the Control Framework using the set_registered_events [Page 483] method Instead, they are registered automatically by the wrapper of the control that you are using.

However, you still have to specify handler methods for the events.

```
DATA tree TYPE REF TO cl_gui_simple_tree.
SET HANDLER dragdrop=>on_drag FOR tree.
```

The events are always registered as system events.

Features

In a drag and drop operation, the Control Framework does not pass any events to the application server until the object is dropped. At the application server, it is separated into up to four standard events that can occur within a drag and drop operation, as described in Process Flow of a Drag and Drop Operation [Page 107]. All events have a drag and drop data object as an event parameter. You use this parameter to manage the context of the drag and drop operation. The particular control wrapper that you are using also provides further information about the drag and drop context. For further information, refer to the documentation of the relevant control wrapper.

- **ONGETFLAVOR**: This event is only triggered if the source and target objects have more than one flavor in common. In the handler method, you must then specify which flavor should be used. To do this, use the set_flavor [Page 506] method on the drag and drop object.
  The event is triggered by the target object of the drag and drop operation.
- **ONDRAG**: This event is triggered when the drag and drop operation is complete at the frontend. When you handle this event, you must determine the context of the target object. You then pass this context to the instance of the class **CL_DRAGDROPOBJECT** that you received as an event parameter.
  The event is triggered by the source object of the drag and drop operation.
- **ONDROP**: When you handle this event, you define what should be done to the target object. To do this, use the event parameter for the context that you filled in the **ONDRAG** event. In this event, you must remember the following:

- Within the ONDROP event, you must make a dynamic typecast. You must catch all possible exceptions of the typecast. In the exception handling you must include handling for the case where you try to assign an invalid object. In this case, you must use the abort [Page 507] method to terminate the drag and drop processing.
- You should select the flavor you want to use so that it is possible to assign the drag and drop object to the right TypeCast.

  The event is triggered by the target object of the drag and drop operation.

- **ONDROPCOMPLETE**: Use this event to perform any further processing necessary after the end of the drag and drop operation. For example, this would be necessary following a move operation.
  The event is triggered by the source object of the drag and drop operation.

Defining Drag and Drop Events in the SAP Tree

This section explains the special considerations that apply to drag and drop operations in the SAP Tree.

Prerequisites

To be able to react to an event, you must first register it. Unlike normal event handling, you do not register drag and drop events with the Control Framework using the set_registered_events [Page 483] method Instead, they are registered automatically by the SAP Tree control wrapper.

However, you still have to specify handler methods for the events.

The events are always registered as system events.

When you fill the node table, you must also specify which nodes are enabled for drag and drop, and the flavors that the nodes should have. You do this by assigning the relevant drag and drop behavior to the field **DRAGDROPID** (see point 3 under Drag and Drop Operations [Page 107]). This requires the following steps (see also the Drag and Drop Programming Example [Page 114]).

4. Define the drag and drop behavior:

```
DATA behaviour_left TYPE REF TO cl_dragdrop.
  CREATE OBJECT behaviour_left.
  CALL METHOD behaviour_left->add
    EXPORTING
        flavor = 'Tree_move_to_Edit'
        dragsrc = 'X'
        droptarget = ' '
        effect = cl_dragdrop=>copy.
```

5. Use the get_handle [Page 501] method to return a handle to the drag and drop behavior:

```
  CALL METHOD behaviour_left->get_handle
     IMPORTING handle = handle_tree.
```

6. Assign the handle to the **DRAGDROPID** field of the corresponding entry in the node table:

```
 node-dragdropid = handle_tree.      " handle of behaviour
```

⚠

Entries with the type **tree->item_class_checkbox** (checkboxes), **tree->item_class_button** (pushbuttons) and **tree->item_class_link** (links) cannot be the source object of a drag and drop operation.

Features

The following table contains the events used in drag and drop:

Event	Description
ON_DROP_GET_FLAVOR	See the event **ONGETFLAVOR** under Drag and Drop Events [Page 109]

ON_DRAG	See the event **ONDRAG** under Drag and Drop Events [Page 109] For trees without multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_SINGLE**).
ON_DRAG_MULTIPLE	See the event **ONDRAG** under Drag and Drop Events [Page 109] For trees with multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_MULTIPLE**).
ON_DROP	See the event **ONDROP** under Drag and Drop Events [Page 109]
ON_DROP_COMPLETE	See the event **ONDROPCOMPLETE** under Drag and Drop Events [Page 109] For trees without multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_SINGLE**).
ON_DROP_COMPLETE_MULTIPLE	See the event **ONDROPCOMPLETE** under Drag and Drop Events [Page 109] For trees with multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_MULTIPLE**).

The individual events have the following parameters:

Event	**Event parameter**	**Description**
ON_DROP_GET_FLAVOR	NODE_KEY	Technical name of the node onto which the source object was dragged
	DRAG_DROP_OBJECT	Data object describing the source object
	FLAVORS	Shared flavors of the drag and drop operation
ON_DRAG	NODE_KEY	Technical name of the node selected as the source object
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DRAG_MULTIPLE	NODE_KEY_TABLE	Table of nodes selected as source obejcts

	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP	NODE_KEY	Technical name of the node onto which the source object was dragged
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP_COMPLETE	NODE_KEY	Technical name of the node selected as the source object
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP_COMPLETE_MULTIPLE	NODE_KEY_TABLE	Table of nodes selected as source obejcts
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object

Example of Drag and Drop Programming

This example program uses a SAP Simple Tree Control and a SAP Textedit Control. The aim is to enable the user to move texts from the tree control into the textedit control.

The example has the program name **RSDEMO_DRAG_DROP_EDIT_TREE**.

```
*&-------------------------------------------------------------------*
*& Report  RSDEMO_DRAG_DROP_EDIT_TREE                              *&
*--------------------------------------------------------------------*
REPORT  rsdemo_drag_drop_edit_tree    .
DATA ok_code TYPE sy-ucomm.
DATA node_itab LIKE node_str OCCURS 0.
DATA node LIKE node_str.
DATA container TYPE REF TO cl_gui_custom_container.
DATA splitter TYPE REF TO cl_gui_easy_splitter_container.
DATA right TYPE REF TO cl_gui_container.
DATA left  TYPE REF TO cl_gui_container.
DATA editor TYPE REF TO cl_gui_textedit.
DATA tree TYPE REF TO cl_gui_simple_tree.
DATA behaviour_left TYPE REF TO cl_dragdrop.
DATA behaviour_right TYPE REF TO cl_dragdrop.
DATA handle_tree TYPE i.
*--------------------------------------------------------------------*
*       CLASS lcl_treeobject DEFINITION
*  container class for drag object
*--------------------------------------------------------------------*
CLASS lcl_drag_object DEFINITION.
  PUBLIC SECTION.
    DATA text TYPE mtreesnode-text.
ENDCLASS.
*---------------------------------------------------------------------*
*       CLASS dragdrop_receiver DEFINITION
*  event handler class for drag&drop events
*---------------------------------------------------------------------*
CLASS lcl_dragdrop_receiver DEFINITION.
  PUBLIC SECTION.
    METHODS:
       flavor_select FOR EVENT on_get_flavor OF cl_gui_textedit
                      IMPORTING index line pos flavors dragdrop_object,
       left_drag FOR EVENT on_drag OF cl_gui_simple_tree
                      IMPORTING node_key drag_drop_object,
       right_drop FOR EVENT ON_DROP OF cl_gui_textedit
                      IMPORTING index line pos dragdrop_object,
       drop_complete FOR EVENT on_drop_complete OF cl_gui_simple_tree
                      IMPORTING node_key drag_drop_object.
ENDCLASS.
START-OF-SELECTION.
  CALL SCREEN 100.
*&-------------------------------------------------------------------*
*&      Module  START  OUTPUT
*&-------------------------------------------------------------------*
MODULE start OUTPUT.
```

```
  SET PF-STATUS 'BASE'.
  IF container is initial.
    CREATE OBJECT container
        EXPORTING container_name = 'CONTAINER'.
    CREATE OBJECT splitter
        EXPORTING parent = container
                  orientation = 1.
    left = splitter->top_left_container.
    right = splitter->bottom_right_container.
    CREATE OBJECT editor
        EXPORTING parent = right.
    CREATE OBJECT tree
        EXPORTING parent = left
                  node_selection_mode = tree->node_sel_mode_single.
* Definition of drag drop behaviour for tree
    CREATE OBJECT behaviour_left.
    CALL METHOD behaviour_left->add
        EXPORTING
              flavor = 'Tree_move_to_Edit'
              dragsrc = 'X'
              droptarget = ' '
              effect = cl_dragdrop=>copy.
   CALL METHOD behaviour_left->add
        EXPORTING
              flavor = 'Tree_copy_to_Edit'
              dragsrc = 'X'
              droptarget = ' '
              effect = cl_dragdrop=>copy.
    CALL METHOD behaviour_left->get_handle
         IMPORTING handle = handle_tree.
* Drag Drop behaviour of tree control nodes are defined in the node
* structure
    PERFORM fill_tree.
    CALL METHOD tree->add_nodes
         EXPORTING node_table = node_itab
                   table_structure_name = 'NODE_STR'.
* Definition of drag drop behaviour for tree
    CREATE OBJECT behaviour_right.
 CALL METHOD behaviour_right->add
        EXPORTING
              flavor = 'Tree_move_to_Edit'
              dragsrc = ' '
              droptarget = 'X'
              effect = cl_dragdrop=>copy.
 CALL METHOD behaviour_right->add
        EXPORTING
              flavor = 'Tree_copy_to_Edit'
              dragsrc = ' '
              droptarget = 'X'
              effect = cl_dragdrop=>copy.
    CALL METHOD editor->set_dragdrop
         EXPORTING dragdrop = behaviour_right.
```

```
* registration of drag and drop events

    SET HANDLER dragdrop=>flavor_select FOR editor.
    SET HANDLER dragdrop=>left_drag FOR tree.
    SET HANDLER dragdrop=>right_drop FOR editor.
    SET HANDLER dragdrop=>drop_complete for TREE.
  ENDIF.
ENDMODULE.                             " START  OUTPUT
*&-------------------------------------------------------------------*
*&      Module  EXIT  INPUT
*&-------------------------------------------------------------------*
MODULE exit INPUT.
  LEAVE PROGRAM.
ENDMODULE.                             " EXIT  INPUT
*&-------------------------------------------------------------------*
*&      Form  fill_tree
*&-------------------------------------------------------------------*
FORM fill_tree.
  DATA: node LIKE mtreesnode.
  CLEAR node.
  node-node_key = 'Root'.
  node-isfolder = 'X'.
  node-text = 'Text'.
  node-dragdropid = ' '.
  APPEND node TO node_itab.
  CLEAR node.
  node-node_key = 'Child1'.
  node-relatkey = 'Root'.
  node-relatship = cl_gui_simple_tree=>relat_last_child.
  node-text = 'DragDrop Text 1'.
  node-dragdropid = handle_tree.       " handle of behaviour
  APPEND node TO node_itab.
  CLEAR node.
  node-node_key = 'Child2'.
  node-relatkey = 'Root'.
  node-relatship = cl_gui_simple_tree=>relat_last_child.
  node-text = 'DragDrop Text 2'.
  node-dragdropid = handle_tree.       " handle of behaviour
  APPEND node TO node_itab.
ENDFORM.                               " fill_tree
*&-------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&-------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CALL METHOD cl_gui_cfw=>dispatch.
ENDMODULE.                             " USER_COMMAND_0100  INPUT
*--------------------------------------------------------------------*
*       CLASS DRAGDROP_RECEIVER IMPLEMENTATION
*--------------------------------------------------------------------*
CLASS lcl_dragdrop_receiver IMPLEMENTATION.
  METHOD flavor_select. " set the right flavor
    IF line > 5.
      SEARCH flavors FOR 'Tree_move_to_Edit'.
      IF sy-subrc = 0.
        CALL METHOD dragDROP_OBJECT->SET_FLAVOR

             EXPORTING newflavor = 'Tree_move_to_Edit'.
      ELSE.
        CALL METHOD dragdrop_object->abort.
      ENDIF.
    ELSE.
      SEARCH flavors FOR 'Tree_copy_to_Edit'.
      IF sy-subrc = 0.
        CALL METHOD dragdrop_object->set_flavor
             EXPORTING newflavor = 'Tree_copy_to_Edit'.
      ELSE.
        CALL METHOD dragdrop_object->abort.
      ENDIF.
    ENDIF.
  ENDMETHOD.
  METHOD left_drag. " define drag object
    DATA drag_object TYPE REF TO lcl_drag_object.
    READ TABLE node_itab WITH KEY node_key = node_key
                         INTO node.
    CREATE OBJECT drag_object.
    drag_object->text = node-text.
    drag_drop_object->object = drag_object.
ENDMETHOD.
  METHOD right_drop. " action in the drop object
    DATA textline(256).
    DATA text_table LIKE STANDARD TABLE OF textline.
    DATA drag_object TYPE REF TO lcl_drag_object.
    CATCH SYSTEM-EXCEPTIONS move_cast_error = 1.
      drag_object ?= dragdrop_object->object.
    ENDCATCH.
    IF sy-subrc = 1.
      " data object has unexpected class
                                   " => cancel Drag & Drop operation
      CALL METHOD dragdrop_object->abort.
      EXIT.
    ENDIF.
    CALL METHOD editor->get_text_as_stream
         IMPORTING text        = text_table.
* Synchronize Automation Queue after Get Methods
    CALL METHOD cl_gui_cfw=>flush.
    textline = drag_object->text.
* Insert text in internal table
    INSERT textline INTO text_table INDEX 1.
* Send modified table to frontend
    CALL METHOD editor->set_text_as_stream
         EXPORTING  text = text_table
         EXCEPTIONS error_dp        = 1
                    error_dp_create = 2.
  ENDMETHOD.
  METHOD drop_complete. " do something after drop
    IF drag_drop_object->flavor = 'Tree_move_to_Edit'.
```

```
      CALL METHOD tree->delete_node
           EXPORTING node_key = node_key.
      delete node_itab where node_key = node_key.

    ENDIF.
    ENDMETHOD.
ENDCLASS.
```

The List Tree

Definition

You create a list tree with reference to the class **cl_gui_list_tree**:

DATA list_tree TYPE REF TO cl_gui_list_tree.

According to the inheritance hierarchy, you can now access the methods of the following classes:

- **cl_gui_object** and **cl_gui_control** (see Methods of the ABAP Objects Control Framework [Page 471]).
- **cl_tree_control_base** (see Methods of the class CL_TREE_CONTROL_BASE [Page 119]).
- **cl_item_control_base** (see Methods of the class CL_ITEM_TREE_CONTROL [Page 157]).
- **cl_gui_list_tree** (see Methods of Class CL_GUI_LIST_TREE [Page 188]).

Use

The program **saptlist_tree_control_demo** demonstrates how to use the list tree.

For details of the attributes of the list tree, refer to the Overview of Tree Classes [Page 18].

Creating a Control: SAP Picture Example

Prerequisites

The following process applies to all SAP custom controls. The programming examples use the SAP Picture Control. However, to apply the example to other controls, you would only have to change the name of the control class.

The example also assumes that you are using the custom control in a Custom Container. The SAP Container documentation contains details of further scenarios.

Process Flow

Create the Instance

17. Define a reference variable for the Custom Container in which you want to place the custom control (**see** SAP Container [Ext.]).

```
DATA container TYPE REF TO cl_gui_custom_container.
```

18. Define a reference variable for the SAP Picture:

```
DATA picture TYPE REF TO cl_gui_picture.
```

19. Create the Custom Container. You must already have created the area **'CUSTOM'** for the Custom Container in the Screen Painter. When you create the container, you must also specify its lifetime [Ext.] (see constructor [Ext.]).

```
CREATE OBJECT container
   EXPORTING container_name = 'CUSTOM'

          lifetime       = lifetime.
```

20. Create the SAP Picture Control. You can also specify a lifetime for the SAP Picture, but it must not be longer than that of its container.

```
CREATE OBJECT picture
    EXPORTING parent  = container

          lifetime = lifetime.
```

Register the Events

21. There are three steps: Registering the events with the Control Framework, defining a handler method, and registering the hander method. These steps are explained under Registering and Processing Events [Page 99].

Use the Control

22. These steps are control-specific and therefore not described here.

Destroy the Control

The lifetime management [Ext.] is normally responsible for destroying any controls you use. However, the following two steps allow you to destroy the control yourself:

23. Use the method free [Page 480] to destroy the Custom Control at the frontend. If you no longer need the control container, release it as well:

```
CALL METHOD picture->free
     EXCEPTIONS cntl_error        = 1
                cntl_system_error = 2.
CALL METHOD container->free
     EXCEPTIONS cntl_error        = 1
                cntl_system_error = 2.
```

⚠

Pay careful attention to the sequence in which you destroy controls at the frontend. When you destroy a container, all controls in it are automatically destroyed as well. If you have already destroyed a control and try to destroy it again, an error occurs. You can check whether a control has already been destroyed using the method is_alive [Page 485].

24. Delete the reference variables to the custom control and the control container.

```
FREE PICTURE.
FREE CONTAINER.
```

Using the List Tree

This section lists the functions that are specific to the list tree.

Prerequisites

The process described here is an extension of the general process for using controls [Page 90] that is specific to the list tree. It does not contain all of the steps required to produce a valid instance of the control.

Process Flow

The program extracts are examples that do not necessarily illustrate all of the features of the control. For precise information, refer to the reference section of this documentation.

Creating the Instance

19. Define a reference variable for the list tree:

```
DATA list_tree TYPE REF TO cl_gui_list_tree.
```

20. If you want to create a heading for the tree, you must create a work area for the hierarchy heading with reference to the structure **treev_hhdr** and one for the list heading with reference to the structure **treev_lhdr**:

```
DATA hierarchy_header TYPE treev_hhdr.
DATA list_header type treev_lhdr.
```

21. Fill the work area for the hierarchy heading. You can set the width (**width** and **width_pix**), the text (**heading**), an icon (**t_image**) and a tool tip (**tooltip**). There are also methods that allow you to change these attributes later on.

```
hierarchy_header-heading = 'Title'.
hierarchy_header-width = 30.
```

22. Fill the work area for the list heading. You can set the text (**heading**), an icon (**t_image**) and a tool tip (**tooltip**).

```
list_header-heading = 'List heading'.
```

23. Create an instance [Page 189] of the SAP Tree:

```
CREATE OBJECT list_tree
    EXPORTING  parent              = container
            node_selection_mode = node_selection_mode
            item_selection      = item_selection
            with_headers        = with_headers
            hierarchy_header    = hierarchy_header
            list_header         = list_header
    EXCEPTIONS lifetime_error      =  1
            cntl_system_error   = 2
            create_error        = 3
```

```
illegal_node_selection_mode = 4
failed                      = 5.
```

Register the Events

24. Register the events [Page 101] for the list tree. The control supports the following events:

Event name	Description
NODE_DOUBLE_CLICK	User double-clicked a node
EXPAND_NO_CHILDREN	User expanded a node that has no children
SELECTION_CHANGED	Selected node has changed
NODE_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	User selected an entry from the context menu
DEFAULT_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an empty space in the control
DEFAULT_CONTEXT_MENU_SELECT	User selected an entry from the context menu
HEADER_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on the heading
HEADER_CONTEXT_MENU_SELECT	User selected an entry from the context menu
ITEM_KEYPRESS	User pressed a key while an entry was selected.
NODE_KEYPRESS	User pressed a key while an entry was selected.
HEADER_CLICK	User clicked a heading

If you set the parameter `item_selection = 'X'` when you created the instance, you can also react to the following events:

Event name	Description
BUTTON_CLICK	The user clicked an item with the class BUTTON
LINK_CLICK	The user clicked an item with the class LINK
CHECKBOX_CHANGE	The user clicked an item with the class CHECKBOX
ITEM_DOUBLE_CLICK	The user double-clicked an item
ITEM_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an item
ITEM_CONTEXT_MENU_SELECT	User selected an entry from the context menu for an item

Using the Column Tree

25. Insert nodes in the tree. To do this, first fill a node table and an item table, then pass them to the control using the add_nodes_and_items [Page 158] method.

```
CALL METHOD list_tree->add_nodes_and_items
   EXPORTING  node_table           = node_table
          item_table            = item_table
          item_table_structure_name = item_table_structure_name
   EXCEPTIONS failed                          = 1
          cntl_system_error             = 2
          error_in_tables               = 3
          dp_error                      = 4
          table_structure_name_not_found = 5.
```

Change existing nodes in the tree, or change the tree attributes (see Changing the Attribtues of the Control [Page 95]).

Query any necessary attributes of the tree and its nodes (see Finding Out the Attributes of the Control [Page 98]).

Destroying the Control

26. Destroy the custom control at the frontend. If you no longer need the control container, release it as well:

```
CALL METHOD list_tree->free.
```

➡

If you are working with the lifetime management [Ext.], you do not need to worry about destroying the control at the frontend yourself. It is done automatically by the system instead.

27. Delete the reference variables to the simple tree and the control container.

```
FREE list_tree.
```

Changing the Attributes of the Control

This section lists all of the methods you can use to change the list tree.

Inserting, Changing, and Deleting Items (With Nodes)

Method	Description
add_nodes_and_items [Page 158]	Adds a set of items (and their nodes)
delete_all_items_of_nodes [Page 162]	Deletes all items for a list of nodes
delete_items [Page 163]	Deletes a set of items
update_nodes_and_items [Page 175]	Changes a list of items (and their nodes)

Changing Individual Items

Method	Description
item_set_chosen [Page 165]	Sets a checkbox in the tree to selected
item_set_disabled [Page 166]	Deactivates an entry in the table
item_set_editable [Page 167]	Sets whether a checkbox can be changed
item_set_font [Page 168]	Sets the font for the item
item_set_hidden [Page 169]	Makes an item invisible
item_set_style [Page 170]	Sets the style of an item
item_set_text [Page 171]	Changes the text of an item
item_set_t_image [Page 172]	Changes the icon of an item
item_set_alignment [Page 199]	Sets the alignment of the item
item_set_length [Page 200]	Sets the displayed length of the item

Selecting a Single Item

Method	Description
select_item [Page 173]	Selects a single item

Expanding Nodes

Method	Description
expand_node [Page 128]	Expands a particular node
expand_nodes [Page 129]	Expands a set of nodes
expand_root_nodes [Page 130]	Expands all root nodes

Selecting Nodes

Method	Description
set_selected_node [Page 152]	Selects a particular node

Changing the Attributes of the Control

select_nodes [Page 147]	Selects a list of nodes
unselect_all [Page 154]	Deselects all seleceted nodes and items
unselect_nodes [Page 155]	Deselects a set of nodes

Deleting Nodes

Method	Description
delete_all_nodes [Page 124]	Deletes all nodes from the tree
delete_node [Page 125]	Deletes a particular node from the tree
delete_nodes [Page 126]	Deletes a set of nodes from the tree

Changing the Attributes of a Node

Method	Description
node_set_disabled [Page 136]	Deactivates nodes
node_set_expander [Page 138]	Sets the **expander** attribute.
node_set_exp_image [Page 139]	Sets expanded node icon
node_set_hidden [Page 140]	Hides a node
node_set_is_folder [Page 141]	Sets the **is_folder** attribute
node_set_no_branch [Page 142]	Sets whether the hierarchy line is drawn to the node
node_set_n_image [Page 143]	Sets the non-expanded node icon
node_set_style [Page 144]	Sets the style of the node
node_set_last_hierarchy_item [Page 191]	Specifies the last item of a node that appears under the hierarchy heading
node_set_dragdropid [Page 137]	Sets the drag and drop behavior of a node

Changing the Attributes of the Hierarchy Heading

Method	Description
hierarchy_header_set_t_image [Page 192]	Changes the icon of the hierarchy heading
hierarchy_header_set_text [Page 194]	Changes the text of the hieararchy heading
hierarchy_header_set_tooltip [Page 201]	Changes the tooltip of the hierarchy heading
hierarchy_header_set_width [Page 196]	Changes the width of the hierarchy heading
hierarchy_header_adjust_width [Page 198]	Adjusts the width of the hierarchy heading

Changing the Attributes of the List Heading

Method	Description
list_header_set_t_image [Page 193]	Changes the icon of the list heading
list_header_set_text [Page 195]	Changes the text of the list heading
list_header_set_tooltip [Page 202]	Changes the tooltip of the list heading

Configuring Keyboard Events

Method	Description
add_key_stroke [Page 120]	Sets a key to trigger an event
remove_all_key_strokes [Page 145]	Deregisters all keys that were registered to trigger an event

Other Methods

Method	Description
ensure_visible [Page 127]	Ensures that a particular node is visible
move_node [Page 135]	Moves a node
scroll [Page 146]	Scrolls in the tree
set_ctx_menu_select_event_appl [Page 156]	Sets whether the event triggered when the user chooses an entry from a context menu should be an application event or a system event
set_has_3d_frame [Page 150]	Sets the 3D frame
set_screen_update [Page 151]	Controls the visibility of changes
set_top_node [Page 153]	Defines the topmost visible node
set_min_node_height [Page 174]	Sets the minimum height of a node
set_default_drop [Page 148]	Sets the drag and drop behavior for dropping on the background of the SAP Tree
set_folder_show_exp_image [Page 149]	Sets the open folder symbol

Finding Out the Attributes of the Control

This section lists all of the methods you can use to retrieve the attributes of the column tree.

Methods for Retrieving Control Attributes

Method	Description
get_expanded_nodes [Page 131]	Lists all expanded nodes
get_selected_node [Page 132]	Returns the name of the selected node
get_selected_nodes [Page 133]	Lists all selected nodes
get_top_node [Page 134]	Name of the topmost visible node
get_selected_item [Page 164]	Name of the selected item
hiearchy_header_get_width [Page 197]	Width of the hierarchy heading

Registering and Processing Events

Purpose

The event mechanism of the Control Framework allows you to use handler methods in your programs to react to events triggered by the control (for example, a double-click).

Prerequisites

The following description has been generalized to apply to all custom controls. For more information specific to a particular control, refer to that control's documentation.

Process Flow

21. Assume you are working with a custom control that has the ABAP wrapper **`cl_gui_xyz`**.

```
DATA my_control TYPE REF TO cl_gui_xyz.
```

Registering Events with the Control Framework

22. Define an internal table (type **`cntl_simple_events`**) and a corresponding work area (type **`cntl_simple_event`**).

```
DATA events TYPE cntl_simple_events.
DATA wa_events TYPE  cntl_simple_event.
```

23. Now fill the event table with the relevant events. To do this, you need the event ID (**`event_id`** field). You can find this information in the Class Browser by looking at the attributes of the class **`cl_gui_xyz`**. You must also decide whether the event is to be a system event (**`appl_event = ' '`**) or an application event (**`appl_event = 'X'`**).

```
wa_events-eventid = event_id.
wa_events-appl_event = appl_event.
APPEND wa_events TO events.
```

24. You must now send the event table to the frontend so that it knows which events it has to direct to the backend.

```
CALL METHOD my_control->set_registered_events
        events = events.
```

To react to the events of you custom control, you must now specify a handler method for it. This can be either an instance method or a static method.

Processing an Event Using an Instance Method

25. Define the (local) class definition for the event handler. To do this, specify the name of the handler method (**`Event_Handler`**). You need to look at the class for the custom control cl_gui_xyz in the Class Browser to find out the name of the event (**`event_name`**) and its parameters (**`event_parameter`**). There is also a default event parameter **`sender`**, which is passed by all events. This contains the reference to the control that triggered the event.

```
CLASS lcl_event_receiver DEFINITION.
PUBLIC SECTION.
METHODS Event_Handler
  FOR EVENT event_name OF cl_gui_xyz
  IMPORTING event_parameter
          sender.
ENDCLASS.
```

26. Register the handler methods with the ABAP Objects Control Framework for the events.

```
DATA event_receiver TYPE REF TO lcl_event_receiver.
CREATE OBJECT event_receiver.
SET HANDLER event_receiver->Event_Handler
        FOR my_control.
```

Processing an Event Using a Static Method

27. Define the (local) class definition for the event handler. To do this, specify the name of the handler method (**`Event_Handler`**). You need to look at the class for the custom control cl_gui_xyz in the Class Browser to find out the name of the event (**`event_name`**) and its parameters (**`event_parameter`**).

```
CLASS lcl_event_receiver DEFINITION.
PUBLIC SECTION.
CLASS-METHODS Event_Handler
      FOR EVENT event_name OF cl_gui_xyz
      IMPORTING event_parameter
            sender.
ENDCLASS.
```

28. Register the handler methods with the ABAP Objects Control Framework for the events.

```
SET HANDLER lcl_event_receiver=>Event_Handler
        FOR my_control.
```

Processing Control Events

29. You define how you want the system to react to an event in the implementation of the handler method.

```
CLASS lcl_event_receiver IMPLEMENTATION.
METHOD Event_Handler.
* Event processing
ENDMETHOD
ENDCLASS.
```

30. If you registered your event as an application event, you need to process it using the method **`CL_GUI_CFW=>DISPATCH`**. For further information, refer to Event Handling [Ext.].

Events of the Column Tree and List Tree

Use

Certain user actions on the column tree and list tree trigger events:

Event	**Event ID** CL_ITEM_TREE_CONTROL=>	**Description**
NODE_DOUBLE_CLICK	EVENTID_NODE_DOUBLE_CLICK	Double-click on a node
NODE_KEYPRESS	EVENTID_NODE_KEYPRESS	The user pressed a key while a node was selected
EXPAND_NO_CHILDREN	EVENTID_EXPAND_NO_CHILDREN	User expanded a node that has no children
SELECTION_CHANGED	EVENTID_SELECTION_CHANGED	You can only use this event if you specified single node selection and **`ITEM_SELECTION = ' '`** when you created the control. Selected node has changed **Important: If you use this event, you cannot use the NODE_DOUBLE_CLICK event.**
NODE_CONTEXT_MENU_REQUEST	EVENTID_NODE_CONTEXT_MENU_REQ	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **`NODE_CONTEXT_MENU_REQUEST`**.	User selected an entry in the context menu for a node
DEFAULT_CONTEXT_MENU_REQUEST	EVENTID_DEF_CONTEXT_MENU_REQ	User requested a context menu with the cursor positioned on the tree background
DEFAULT_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **`DEFAULT_CONTEXT_MENU_REQUEST`**.	User selected an entry from the context menu for the tree background
HEADER_CONTEXT_MENU_REQUEST	EVENTID_HEADER_CONTEXT_MEN_REQ	User requested a context menu with the cursor positioned on a heading

HEADER_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **`EVENTID_HEADER_CONTEXT_MEN_REQ.`**	User selected an entry from the context menu for the tree background
HEADER_CLICK	EVENTID_HEADER_CLICK	User clicked a heading
ON_DROP_GET_FLAVOR	See Drag and Drop Events in the SAP Tree [Page 111]	There are several different drag and drop flavors
ON_DRAG	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the source object (single selection)
ON_DRAG_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the source object (multiple selection)
ON_DROP	See Drag and Drop Events in the SAP Tree [Page 111]	Determines the context in the target object
ON_DROP_COMPLETE	See Drag and Drop Events in the SAP Tree [Page 111]	Last event before completion of the drag and drop (single selection)
ON_DROP_COMPLETE_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	Last event before completion of the drag and drop (multiple selection)

If you set the parameter **`item_selection = 'X'`** when you created the instance, you can also react to the following events:

Event	**Event ID** CL_ITEM_TREE_CONTROL=>	**Description**
ITEM_DOUBLE_CLICK	EVENTID_ITEM_DOUBLE_CLICK	The user double-clicked an item
ITEM_KEYPRESS	EVENTID_ITEM_KEYPRESS	The user pressed a key while a node was selected
BUTTON_CLICK	EVENTID_BUTTON_CLICK	The user clicked an item with type BUTTON
LINK_CLICK	EVENTID_LINK_CLICK	The user clicked an item with type LINK
CHECKBOX_CHANGE	EVENTID_CHECKBOX_CHANGE	The user clicked an item with type CHECKBOX
ITEM_CONTEXT_MENU_REQUEST	EVENTID_ITEM_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on a node
ITEM_CONTEXT_MENU_SELECT	This event is registered automatically when you register the event **`ITEM_CONTEXT_MENU_REQUEST.`**	User selected an entry from the context menu

Some events also export parameters:

Event	Parameters	Description
NODE_DOUBLE_CLICK	NODE_KEY	Node on which the user double-clicked
NODE_KEYPRESS	NODE_KEY	Node selected when the user pressed the key
	KEY	Key pressed
EXPAND_NO_CHILDREN	NODE_KEY	Node without child nodes that the user tried to expand
SELECTION_CHANGED	NODE_KEY	New selected node
NODE_CONTEXT_MENU_REQUEST	NODE_KEY	Node selected when the user requested the context menu
	MENU	Menu to be displayed (must be filled in the event handler)
NODE_CONTEXT_MENU_SELECT	NODE_KEY	Node selected when the user chose an entry from the context menu
	FCODE	Function code of the selected entry in the context menu
HEADER_CLICK	HEADER_NAME	Name of the heading clicked by the user
HEADER_CONTEXT_MENU_REQUEST	HEADER_NAME	Heading selected when the user requested the context menu
	MENU	Menu to be displayed (must be filled in the event handler)
HEADER_CONTEXT_MENU_SELECT	HEADER_NAME	Heading selected when the user selected from the context menu
	FCODE	Function code of the selected entry in the context menu
CHECKBOX_CHANGE	NODE_KEY	Name of the node
	ITEM_NAME	Name of the item clicked by the user
	CHECKED	'X': **Checkbox selected** ' ': **Checkbox not selected**
ITEM_DOUBLE_CLICK	NODE_KEY	Name of the node

	ITEM_NAME	Name of the item clicked by the user
ITEM_CONTEXT_MENU_REQUEST	NODE_KEY	Name of the node
	ITEM_NAME	Name of the item on which the context menu was request
	MENU	Menu to be displayed (must be filled in the event handler)
ITEM_CONTEXT_MENU_SELECT	NODE_KEY	Name of the node
	ITEM_NAME	Name of the item on which the context menu was request
	FCODE	Function code of the selected entry in the context menu
ITEM_KEYPRESS	NODE_KEY	Name of the node
	ITEM_NAME	Name of the item selected when the user pressed a key
	KEY	Key pressed
DEFAULT_CONTEXT_MENU_REQUEST	MENU	Menu to be displayed (must be filled in the event handler)
DEFAULT_CONTEXT_MENU_SELECT	FCODE	Function code of the selected entry in the context menu
ON_DROP_GET_FLAVOR	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DRAG	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DRAG_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DROP	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DROP_COMPLETE	See Drag and Drop Events in the SAP Tree [Page 111]	
ON_DROP_COMPLETE_MULTIPLE	See Drag and Drop Events in the SAP Tree [Page 111]	

If you want to use events that rely on the user pressing a key (for example, **`NODE_KEYPRESS`**), you must register the keystroke using the method add_key_stroke [Page 120]. You can deregister the registered key strokes using the method remove_all_key_strokes [Page 145].

Integration

To react to an event in your ABAP program, you must have registered it. To do this, use the method set_registered_events [Page 483]. Events that are triggered but for which you are not registered are filtered by the presentation server, and not passed to the application server. **See** event handling [Ext.].

Activities

Read the general process [Page 90] for working with events in the Control Framework.

Drag and Drop

Use

Drag and drop allows the user to select an object from one part of a custom control (source) and drop it on another part of a custom control (target). An action occurs in the second part that depends on the object type. Source and target may be either the same control or different controls.

Prerequisites

For a control to support drag and drop, the control wrapper must provide drag and drop events. You must then write handler methods for these events in your program. The events are registered automatically by the relevant control wrapper.

Features

A particular drag and drop behavior is set for each custom control. This behavior may be set globally for all elements of the control (for example, SAP Textedit), or you may be able to define a different behavior for each component (for example SAP Tree). Each behavior consists of one or more descriptions.

A description has the following attributes:

- **DragSrc**: Object is the source of a drag and drop procedure
- **DropTarget**: Object is the target of a drag and drop procedure
- **Flavor**: The flavor describes the type of a drag and drop description. In a drag and drop operation, you can only drop an object onto another if both have at least one common description.
- **Effect**: Specifies whether the drag and drop operations copies or moves the object.
- **Effect_In_Ctrl**: The drop effect used when you copy or move data within the same control.

As soon as a drag event is triggered, you must use the corresponding handler method to find out the affected object.

You must also define the action that is to be carried out on the drop event. The action usually depends on the object that you drop in the control.

If you assign more than one flavor to an object, you must define which flavor is to be used. You do this in the handler for another event.

Once the drop event is finished, you can use a further event to implement additional actions. This is particularly useful for deleting the dropped object from the source after a move operation.

Activities

Whenever you provide a drag and drop function to move objects, you should always provide an *Undo* function as well. You must implement this yourself in the application.

Process Flow of a Drag and Drop Operation

Prerequisites

The following section explains how a drag and drop operation works, examining into the roles of the application server and frontend, and going on to identify the individual steps required to program drag and drop in an application.

Process Flow

Application Server

29. You create the custom control [Page 90].
30. You register the drag and drop events [Page 109].
31. You define the drag and drop behavior for the individual custom controls or their components. To do this, you create an instance [Page 494] of the class CL_DRAGDROP [Page 493]. You then assign one or more flavors [Page 495] to this instance. These describe the drag and drop behavior of the relevant custom control. During the program, you can change [Page 502], delete [Page 504], and query [Page 499] the flavors in your program. You can also initialize [Page 497] or destroy [Page 498] the entire instance.
32. You assign flavors to the custom control using specific methods of the relevant control. For further information, refer to the corresponding control documentation.

Frontend

The following steps are performed by the system at the frontend. They are only listed here so that you can understand what happens during a drag and drop operation.

33. Once the use has selected an object with the left mouse button, the drag and drop service starts.
34. The drag and drop service checks whether a drag and drop behavior has been defined for the object, and whether the object can be dragged (DragSource attribute).
35. If, according to the DragSource attribute, the object can be dragged, the drag and drop operation starts. The mouse pointer then changes automatically.
36. As long as the left mouse button remains pressed, the system continually checks whether the mouse pointer is positioned over an object in a custom control that can receive a dropped object (DropTarget attribute), and whether the flavor of that object is the same as the flavor of the source. If this is the case, the mouse pointer changes again to inform the user.
37. If the user now drops the object, an event is triggered to inform the application server.

> This concludes the drag and drop operation for the frontend. However, there has not yet been any change to the contents of the custom control.

Application Server

38. The drag and drop service of the application server creates an instance of the class CL_DRAGDROPOBJECT [Page 505]. You can use this instance (for example,

`drag_drop_object`) in all events of the drag and drop process as an event parameter. You can use it to find out the context between the events.

39. The drag and drop service checks whether the drag object and drop object have more than one flavor in common. If this is the case, the event `ONGETFLAVOR` is triggered. In the corresponding handler method, you must decide which flavor to use. You do this using the method set_flavor [Page 506].
40. Now, the drag and drop event `ONDRAG` is triggered. It has event parameters that tell you which object the user has dragged. Within the handler routine, you must pass the context (information about the source object) to the instance of the drag and drop data object created in step 9.
    `drag_drop_object->object = mydragobject.`
41. Next, the `ONDROP` event is triggered. The corresponding handler method serves to process the drag and drop data object. Here, you have to implement the changes that are to be made to the target object based on the drag and drop operation.
42. The last event of the drag and drop operation is `ONDROPCOMPLETE`. This is where you can make your last changes to the drag and drop object. In particular, you should use this event to delete the source object from the DragSource control and the corresponding data structures if you have used the drag and drop operation to move the object.

    ➡

    The Example of Drag and Drop Programming [Page 114] contains an example of a drag and drop operation between a SAP Tree and a SAP Textedit.

Drag and Drop Events

This section only describes those properties of drag and drop events that apply to all controls. The individual control wrappers may augment them. You should therefore consult the relevant control documentation to see if that control has any peculiarities.

Use

There are four standard events in a drag and drop operation at which control is returned to the application program. You use the event handler methods for these events to implement the actions that should be performed during the operation.

> Some control wrappers offer additional drag and drop events. For further information, refer to the documentation of the individual controls.

Prerequisites

To be able to react to an event, you must first register it. Unlike normal event handling, you do not register drag and drop events with the Control Framework using the set_registered_events [Page 483] method Instead, they are registered automatically by the wrapper of the control that you are using.

However, you still have to specify handler methods for the events.

```
DATA tree TYPE REF TO cl_gui_simple_tree.
SET HANDLER dragdrop=>on_drag FOR tree.
```

The events are always registered as system events.

Features

In a drag and drop operation, the Control Framework does not pass any events to the application server until the object is dropped. At the application server, it is separated into up to four standard events that can occur within a drag and drop operation, as described in Process Flow of a Drag and Drop Operation [Page 107]. All events have a drag and drop data object as an event parameter. You use this parameter to manage the context of the drag and drop operation. The particular control wrapper that you are using also provides further information about the drag and drop context. For further information, refer to the documentation of the relevant control wrapper.

- **ONGETFLAVOR**: This event is only triggered if the source and target objects have more than one flavor in common. In the handler method, you must then specify which flavor should be used. To do this, use the set_flavor [Page 506] method on the drag and drop object.
  The event is triggered by the target object of the drag and drop operation.

- **ONDRAG**: This event is triggered when the drag and drop operation is complete at the frontend. When you handle this event, you must determine the context of the target object. You then pass this context to the instance of the class **CL_DRAGDROPOBJECT** that you received as an event parameter.
  The event is triggered by the source object of the drag and drop operation.

- **ONDROP**: When you handle this event, you define what should be done to the target object. To do this, use the event parameter for the context that you filled in the **ONDRAG** event. In this event, you must remember the following:

   - Within the ONDROP event, you must make a dynamic typecast. You must catch all possible exceptions of the typecast. In the exception handling you must include handling for the case where you try to assign an invalid object. In this case, you must use the abort [Page 507] method to terminate the drag and drop processing.
   - You should select the flavor you want to use so that it is possible to assign the drag and drop object to the right TypeCast.

     The event is triggered by the target object of the drag and drop operation.

- **ONDROPCOMPLETE**: Use this event to perform any further processing necessary after the end of the drag and drop operation. For example, this would be necessary following a move operation.
  The event is triggered by the source object of the drag and drop operation.

Defining Drag and Drop Events in the SAP Tree

This section explains the special considerations that apply to drag and drop operations in the SAP Tree.

Prerequisites

To be able to react to an event, you must first register it. Unlike normal event handling, you do not register drag and drop events with the Control Framework using the set_registered_events [Page 483] method Instead, they are registered automatically by the SAP Tree control wrapper.

However, you still have to specify handler methods for the events.

The events are always registered as system events.

When you fill the node table, you must also specify which nodes are enabled for drag and drop, and the flavors that the nodes should have. You do this by assigning the relevant drag and drop behavior to the field **DRAGDROPID** (see point 3 under Drag and Drop Operations [Page 107]). This requires the following steps (see also the Drag and Drop Programming Example [Page 114]).

7. Define the drag and drop behavior:

```
DATA behaviour_left TYPE REF TO cl_dragdrop.
  CREATE OBJECT behaviour_left.
  CALL METHOD behaviour_left->add
    EXPORTING
        flavor = 'Tree_move_to_Edit'
        dragsrc = 'X'
        droptarget = ' '
        effect = cl_dragdrop=>copy.
```

8. Use the get_handle [Page 501] method to return a handle to the drag and drop behavior:

```
  CALL METHOD behaviour_left->get_handle
     IMPORTING handle = handle_tree.
```

9. Assign the handle to the **DRAGDROPID** field of the corresponding entry in the node table:

```
 node-dragdropid = handle_tree.      " handle of behaviour
```

⚠

Entries with the type **tree->item_class_checkbox** (checkboxes), **tree->item_class_button** (pushbuttons) and **tree->item_class_link** (links) cannot be the source object of a drag and drop operation.

Features

The following table contains the events used in drag and drop:

Event	Description
ON_DROP_GET_FLAVOR	See the event **ONGETFLAVOR** under Drag and Drop Events [Page 109]

ON_DRAG	See the event **ONDRAG** under Drag and Drop Events [Page 109] For trees without multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_SINGLE**).
ON_DRAG_MULTIPLE	See the event **ONDRAG** under Drag and Drop Events [Page 109] For trees with multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_MULTIPLE**).
ON_DROP	See the event **ONDROP** under Drag and Drop Events [Page 109]
ON_DROP_COMPLETE	See the event **ONDROPCOMPLETE** under Drag and Drop Events [Page 109] For trees without multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_SINGLE**).
ON_DROP_COMPLETE_MULTIPLE	See the event **ONDROPCOMPLETE** under Drag and Drop Events [Page 109] For trees with multiple selection (**NODE_SELECTION_MODE = TREE->NODE_SEL_MODE_MULTIPLE**).

The individual events have the following parameters:

Event	Event parameter	Description
ON_DROP_GET_FLAVOR	NODE_KEY	Technical name of the node onto which the source object was dragged
	DRAG_DROP_OBJECT	Data object describing the source object
	FLAVORS	Shared flavors of the drag and drop operation
ON_DRAG	NODE_KEY	Technical name of the node selected as the source object
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DRAG_MULTIPLE	NODE_KEY_TABLE	Table of nodes selected as source obejcts

	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP	NODE_KEY	Technical name of the node onto which the source object was dragged
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP_COMPLETE	NODE_KEY	Technical name of the node selected as the source object
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object
ON_DROP_COMPLETE_MULTIPLE	NODE_KEY_TABLE	Table of nodes selected as source obejcts
	ITEM_NAME **(not in simple tree)**	Technical name of the item selected as the source object
	DRAG_DROP_OBJECT	Data object describing the source object

Example of Drag and Drop Programming

This example program uses a SAP Simple Tree Control and a SAP Textedit Control. The aim is to enable the user to move texts from the tree control into the textedit control.

The example has the program name **RSDEMO_DRAG_DROP_EDIT_TREE**.

```
*&-------------------------------------------------------------------*
*& Report  RSDEMO_DRAG_DROP_EDIT_TREE                              *&
*--------------------------------------------------------------------*
REPORT  rsdemo_drag_drop_edit_tree    .
DATA ok_code TYPE sy-ucomm.
DATA node_itab LIKE node_str OCCURS 0.
DATA node LIKE node_str.
DATA container TYPE REF TO cl_gui_custom_container.
DATA splitter TYPE REF TO cl_gui_easy_splitter_container.
DATA right TYPE REF TO cl_gui_container.
DATA left  TYPE REF TO cl_gui_container.
DATA editor TYPE REF TO cl_gui_textedit.
DATA tree TYPE REF TO cl_gui_simple_tree.
DATA behaviour_left TYPE REF TO cl_dragdrop.
DATA behaviour_right TYPE REF TO cl_dragdrop.
DATA handle_tree TYPE i.
*--------------------------------------------------------------------*
*       CLASS lcl_treeobject DEFINITION
*  container class for drag object
*--------------------------------------------------------------------*
CLASS lcl_drag_object DEFINITION.
  PUBLIC SECTION.
    DATA text TYPE mtreesnode-text.
ENDCLASS.
*---------------------------------------------------------------------*
*       CLASS dragdrop_receiver DEFINITION
*  event handler class for drag&drop events
*---------------------------------------------------------------------*
CLASS lcl_dragdrop_receiver DEFINITION.
  PUBLIC SECTION.
    METHODS:
       flavor_select FOR EVENT on_get_flavor OF cl_gui_textedit
                      IMPORTING index line pos flavors dragdrop_object,
       left_drag FOR EVENT on_drag OF cl_gui_simple_tree
                      IMPORTING node_key drag_drop_object,
       right_drop FOR EVENT ON_DROP OF cl_gui_textedit
                      IMPORTING index line pos dragdrop_object,
       drop_complete FOR EVENT on_drop_complete OF cl_gui_simple_tree
                      IMPORTING node_key drag_drop_object.
ENDCLASS.
START-OF-SELECTION.
  CALL SCREEN 100.
*&-------------------------------------------------------------------*
*&      Module  START  OUTPUT
*&-------------------------------------------------------------------*
MODULE start OUTPUT.
```

```
  SET PF-STATUS 'BASE'.
  IF container is initial.
    CREATE OBJECT container
        EXPORTING container_name = 'CONTAINER'.
    CREATE OBJECT splitter
        EXPORTING parent = container
                  orientation = 1.
    left = splitter->top_left_container.
    right = splitter->bottom_right_container.
    CREATE OBJECT editor
        EXPORTING parent = right.
    CREATE OBJECT tree
        EXPORTING parent = left
                  node_selection_mode = tree->node_sel_mode_single.
* Definition of drag drop behaviour for tree
    CREATE OBJECT behaviour_left.
    CALL METHOD behaviour_left->add
        EXPORTING
              flavor = 'Tree_move_to_Edit'
              dragsrc = 'X'
              droptarget = ' '
              effect = cl_dragdrop=>copy.
   CALL METHOD behaviour_left->add
        EXPORTING
              flavor = 'Tree_copy_to_Edit'
              dragsrc = 'X'
              droptarget = ' '
              effect = cl_dragdrop=>copy.
    CALL METHOD behaviour_left->get_handle
         IMPORTING handle = handle_tree.
* Drag Drop behaviour of tree control nodes are defined in the node
* structure
    PERFORM fill_tree.
    CALL METHOD tree->add_nodes
         EXPORTING node_table = node_itab
                   table_structure_name = 'NODE_STR'.
* Definition of drag drop behaviour for tree
    CREATE OBJECT behaviour_right.
 CALL METHOD behaviour_right->add
        EXPORTING
              flavor = 'Tree_move_to_Edit'
              dragsrc = ' '
              droptarget = 'X'
              effect = cl_dragdrop=>copy.
 CALL METHOD behaviour_right->add
        EXPORTING
              flavor = 'Tree_copy_to_Edit'
              dragsrc = ' '
              droptarget = 'X'
              effect = cl_dragdrop=>copy.
    CALL METHOD editor->set_dragdrop
         EXPORTING dragdrop = behaviour_right.
```

```
* registration of drag and drop events

    SET HANDLER dragdrop=>flavor_select FOR editor.
    SET HANDLER dragdrop=>left_drag FOR tree.
    SET HANDLER dragdrop=>right_drop FOR editor.
    SET HANDLER dragdrop=>drop_complete for TREE.
  ENDIF.
ENDMODULE.                             " START  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  EXIT  INPUT
*&---------------------------------------------------------------------*
MODULE exit INPUT.
  LEAVE PROGRAM.
ENDMODULE.                             " EXIT  INPUT
*&---------------------------------------------------------------------*
*&      Form  fill_tree
*&---------------------------------------------------------------------*
FORM fill_tree.
  DATA: node LIKE mtreesnode.
  CLEAR node.
  node-node_key = 'Root'.
  node-isfolder = 'X'.
  node-text = 'Text'.
  node-dragdropid = ' '.
  APPEND node TO node_itab.
  CLEAR node.
  node-node_key = 'Child1'.
  node-relatkey = 'Root'.
  node-relatship = cl_gui_simple_tree=>relat_last_child.
  node-text = 'DragDrop Text 1'.
  node-dragdropid = handle_tree.       " handle of behaviour
  APPEND node TO node_itab.
  CLEAR node.
  node-node_key = 'Child2'.
  node-relatkey = 'Root'.
  node-relatship = cl_gui_simple_tree=>relat_last_child.
  node-text = 'DragDrop Text 2'.
  node-dragdropid = handle_tree.       " handle of behaviour
  APPEND node TO node_itab.
ENDFORM.                               " fill_tree
*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.
  CALL METHOD cl_gui_cfw=>dispatch.
ENDMODULE.                             " USER_COMMAND_0100  INPUT
*---------------------------------------------------------------------*
*       CLASS DRAGDROP_RECEIVER IMPLEMENTATION
*---------------------------------------------------------------------*
CLASS lcl_dragdrop_receiver IMPLEMENTATION.
  METHOD flavor_select. " set the right flavor
    IF line > 5.
      SEARCH flavors FOR 'Tree_move_to_Edit'.
      IF sy-subrc = 0.
        CALL METHOD dragDROP_OBJECT->SET_FLAVOR

             EXPORTING newflavor = 'Tree_move_to_Edit'.
      ELSE.
        CALL METHOD dragdrop_object->abort.
      ENDIF.
    ELSE.
      SEARCH flavors FOR 'Tree_copy_to_Edit'.
      IF sy-subrc = 0.
        CALL METHOD dragdrop_object->set_flavor
             EXPORTING newflavor = 'Tree_copy_to_Edit'.
      ELSE.
        CALL METHOD dragdrop_object->abort.
      ENDIF.
    ENDIF.
  ENDMETHOD.
  METHOD left_drag. " define drag object
    DATA drag_object TYPE REF TO lcl_drag_object.
    READ TABLE node_itab WITH KEY node_key = node_key
                         INTO node.
    CREATE OBJECT drag_object.
    drag_object->text = node-text.
    drag_drop_object->object = drag_object.
ENDMETHOD.
  METHOD right_drop. " action in the drop object
    DATA textline(256).
    DATA text_table LIKE STANDARD TABLE OF textline.
    DATA drag_object TYPE REF TO lcl_drag_object.
    CATCH SYSTEM-EXCEPTIONS move_cast_error = 1.
      drag_object ?= dragdrop_object->object.
    ENDCATCH.
    IF sy-subrc = 1.
      " data object has unexpected class
                                   " => cancel Drag & Drop operation
      CALL METHOD dragdrop_object->abort.
      EXIT.
    ENDIF.
    CALL METHOD editor->get_text_as_stream
         IMPORTING text        = text_table.
* Synchronize Automation Queue after Get Methods
    CALL METHOD cl_gui_cfw=>flush.
    textline = drag_object->text.
* Insert text in internal table
    INSERT textline INTO text_table INDEX 1.
* Send modified table to frontend
    CALL METHOD editor->set_text_as_stream
         EXPORTING  text = text_table
         EXCEPTIONS error_dp        = 1
                    error_dp_create = 2.
  ENDMETHOD.
  METHOD drop_complete. " do something after drop
    IF drag_drop_object->flavor = 'Tree_move_to_Edit'.
```

```
      CALL METHOD tree->delete_node
           EXPORTING node_key = node_key.
      delete node_itab where node_key = node_key.

    ENDIF.
    ENDMETHOD.
ENDCLASS.
```

Methods of Class CL_TREE_CONTROL_BASE

All SAP Tree classes can use the methods of this class.

add_key_stroke

Use this method to define keys that trigger an event. To react to the events, you must also register the corresponding event (**NODE_KEYPRESS** and/or **ITEM_KEYPRESS**).

```
CALL METHOD tree->add_key_stroke
   EXPORTING  key = key
   EXCEPTIONS failed          = 1
          illegal_key      = 2
          cntl_system_error = 3.
```

Parameters	Description
key	Key that you want to trigger the event: **CL_TREE_CONTROL_BASE=>KEY_F1**: Function key F1 **CL_TREE_CONTROL_BASE=>KEY_F4**: Function key F4 **CL_TREE_CONTROL_BASE=>KEY_INSERT**: Insert key **CL_TREE_CONTROL_BASE=>KEY_DELETE**: Delete key

collapse_all_nodes

This method allows you to collapse the tree from within your program. The result is that only the root nodes are displayed.

```
CALL METHOD tree->collapse_all_nodes
   EXCEPTIONS failed          = 1
          cntl_system_error = 2.
```

collapse_nodes

Use this method to close all the folders specified in the node table.

```
CALL METHOD tree->collapse_nodes
    EXPORTING  node_key_table = node_key_table
    EXCEPTIONS failed              = 1
            cntl_system_error      = 2
            error_in_node_key_table = 3
            dp_error               = 4.
```

Parameters	Description
node_key_table	Node table containing the folders you want to close. Declare the node table with reference to the type **treev_nks**.

collapse_subtree

Use this method to close a specified folder.

```
CALL METHOD tree->collapse_subtree
   EXPORTING node_key = node_key
   EXCEPTIONS failed          = 1
          node_not_found    = 2
          cntl_system_error = 3.
```

Parameters	Description
node_key	Folder you want to close. The parameter is defined with reference to type **`tv_nodekey`**.

delete_all_nodes

Use this method to delete all nodes from the tree.

```
CALL METHOD tree->delete_all_nodes
    EXCEPTIONS failed           = 1
            cntl_system_error = 2.
```

delete_node

Use this method to delete the node **node_key** from the tree. If the node is a folder, all of its child nodes will be deleted as well.

```
CALL METHOD tree->delete_node
    EXPORTING node_key = node_key
    EXCEPTIONS failed           = 1
            node_not_found    = 2
            cntl_system_error = 3.
```

Parameters	Description
node_key	Node you want to delete. The parameter is defined with reference to type **tv_nodekey**.

delete_nodes

Use this method to delete all the nodes specified in the node table **`node_key_table`**.

```
CALL METHOD tree->delete_nodes
   EXPORTING node_key_table = node_key_table
   EXCEPTIONS failed                  = 1
          cntl_system_error      = 2
          error_in_node_key_table = 3
          dp_error               = 4.
```

Parameters	Description
node_key_table	Node table containing the nodes you want to delete. Declare the node table with reference to the type **`treev_nks`**.

If you want to delete a node's child nodes explicitly, you must make sure that you list them in the table before the parent node. However, deleting the parent node is sufficient, since all of its child nodes will be deleted with it.

ensure_visible

Use this method to ensure that a particular node is visible.

```
CALL METHOD tree->ensure_visible
    EXPORTING node_key = node_key
    EXCEPTIONS failed           = 1
             node_not_found    = 2
             cntl_system_error = 3.
```

Parameters	Description
node_key	Node that you want to ensure is visible. The parameter is defined with reference to type **tv_nodekey**.

expand_node

Use this method to expand a particular node.

```
CALL METHOD tree->expand_node
   EXPORTING node_key      = node_key
         level_count    = level_count
         expand_subtree = expand_subtree
   EXCEPTIONS failed            = 1
          illegal_level_count  = 2
          cntl_system_error    = 3
          node_not_found       = 4
          cannot_expand_leaf   = 5.
```

Parameters	Description
node_key	Node you want to expand. The parameter is defined with reference to type **tv_nodekey**.
level_count	Depth to which you want to expand nodes. 0: Only the current node is expanded. 1: The current node and the next hierarchy level are expanded. and so on.
expand_subtree	'**X**': Expands all nodes in the subtree. The system ignores any value of **level_count**.

expand_nodes

Use this method to expand a list of nodes.

```
CALL METHOD tree->expand_nodes
    EXPORTING node_key_table = node_key_table
    EXCEPTIONS failed                  = 1
            cntl_system_error       = 2
            ERROR_IN_NODE_KEY_TABLE = 3
            DP_ERROR                = 4.
```

Parameters	Description
node_key_table	Node table containing the nodes you want to expand. Declare the node table with reference to the type **`treev_nks`**.

expand_root_nodes

Use this method to expand all root nodes.

```
CALL METHOD tree->expand_root_nodes
    EXPORTING level_count    = level_count
            expand_subtree = expand_subtree
    EXCEPTIONS failed              = 1
            illegal_level_count  = 2
            cntl_system_error    = 3.
```

Parameters	Description
level_count	Depth to which you want to expand nodes. 0: Only the root nodes are expanded - no underlying nodes. 1: The current node and the next hierarchy level are expanded. and so on.
expand_subtree	'**X**': Expands all nodes in the subtree. The system ignores any value of LEVEL_COUNT.

get_expanded_nodes

This method returns a node table containing the keys of all expanded nodes.

```
CALL METHOD tree->get_expanded_nodes
   CHANGING   node_key_table = node_key_table
   EXCEPTIONS cntl_system_error      = 1
          dp_error               = 2
          failed                 = 3.
```

Parameters	Description
node_key_table	Node table containing the expanded nodes. Declare the node table with reference to the type **treev_nks**.

get_selected_node

This method returns a selected node.

⚠

You may only use this method with tree controls where only one node may be selected at any one time. (That is, created **`using node_selection_mode = tree->node_sel_mode_single`**.)

```
CALL METHOD tree->get_selected_node
    IMPORTING  node_key       = node_key
    EXCEPTIONS failed                  = 1
            single_node_selection_only = 2
            cntl_system_error          = 3.
```

Parameters	Description
node_key	Node selected in the tree control. The parameter is defined with reference to type **`tv_nodekey`**.

get_selected_nodes

This method returns a node table containing the keys of all selected nodes.

You may only use this method with tree controls where multiple nodes may be selected at any one time. (That is, created using **node_selection_mode = tree->node_sel_mode_multiple**.)

```
CALL METHOD tree->get_selected_nodes
   CHANGING   node_key_table = node_key_table
   EXCEPTIONS cntl_system_error          = 1
          dp_error                   = 2
          failed                     = 3
          multiple_node_selection_only = 4.
```

Parameters	Description
node_key_table	Node table containing the selected nodes. Declare the node table with reference to the type **treev_nks**.

get_top_node

This method returns the topmost node in the display.

```
CALL METHOD tree->get_top_node
    IMPORTING  node_key       = node_key
    EXCEPTIONS failed                 = 1
           cntl_system_error         = 2.
```

Parameters	Description
node_key	Topmost node in the control display The parameter is defined with reference to type **`tv_nodekey`**.

move_node

Use this method to mode nodes within the tree. Subordinate nodes of the node that you move are also moved.

```
CALL METHOD tree->move_node
    EXPORTING  node_key      = node_key
           relatkey       = relatkey
           relatship      = relatship
    EXCEPTIONS failed              = 1
           cntl_system_error    = 2
           node_not_found       = 3
           move_error         = 4
           relative_not_found   = 5
           illegal_relatship    = 6
           parent_is_leaf       = 7.
```

Parameters	Description
node_key	Name of the node you want to move. The parameter is defined with reference to type **tv_nodekey**.
relatkey	Name of the related node.
relatship	Relationship between **node_key** and **relatkey**: **tree->relat_first_child** (**node_key** is first child node of node **relatkey**) **tree->relat_last_child** (**node_key** is the last child node of node **relatkey**) **tree->relat_prev_sibling** (**node_key** is inserted before **relatkey** at the same hierarchy level) **tree->relat_prev_sibling** (**node_key** is inserted after **relatkey** at the same hierarchy level) **tree->relat_first_sibling** (**node_key** is inserted before all other nodes at the same hierarchy level as **relatkey**). **tree->relat_last_sibling** (**node_key** is inserted after all other nodes at the same hierarchy level as **relatkey**).

⚠

You cannot reassign a node below one of its children.

node_set_disabled

Use this method to deactivate a node. These nodes cannot then be selected. Furthermore, no other actions, such as double-clicking, are possible.

```
CALL METHOD tree->node_set_disabled
   EXPORTING  node_key       = node_key
          disabled       = disabled
   EXCEPTIONS failed              = 1
          node_not_found       = 2
          cntl_system_error    = 3.
```

Parameters	Description
node_key	Name of the node that you want to deactivate. The parameter is defined with reference to type **tv_nodekey**.
disabled	'**X**': Deactivate the node ' ': Activate the node

node_set_dragdropid

Use this method to set the drag and drop behavior of a node.

```
CALL METHOD tree->node_set_dragdropid
    EXPORTING  node_key       = node_key
            dragdropid     = dragdropid
    EXCEPTIONS failed              = 1
            node_not_found      = 2
            cntl_system_error    = 3.
```

Parameters	Description
node_key	Name of the node that you want to deactivate. The parameter is defined with reference to type **tv_nodekey**.
dragdropid	Drag and drop [Page 106] behavior that you want to assign to the node

node_set_expander

You may only assign the **expander** attribute to nodes for which the **isfolder** attribute has also been set. These nodes are also called folders. When you set the **expander** attribute, a plus sign appears next to the folder (so that you can expand it), even if the folder is currently empty. If the user expands an empty branch, the control triggers the event **EXPAND_NO_CHILDREN** .

Setting this attribute is useful if you only want to send data to the tree control on request. In this case, you can set the **expander** attribute for those nodes under which further information might be requested. If a user expands one of these nodes, the **EXPAND_NO_CHILDREN** event is triggered, and you can pass the relevant information back to the tree control in the corresponding event handler method.

Prerequisites

The **is_folder** attribute must be set for the node in question.

```
CALL METHOD tree->node_set_expander
     EXPORTING  node_key  = node_key
                expander  = expander
     EXCEPTIONS failed                = 1
                node_not_found        = 2
                cntl_system_error     = 3.
```

Parameters	Description
node_key	Name of the node for which you want to set the **expander** attribute. The parameter is defined with reference to type **tv_nodekey**.
expander	'**X**': Sets the **expander** attribute for the node. ' ': Does not set the **expander** attribute for the node.

node_set_exp_image

Use this method to set the symbol that denotes an open folder.

```
CALL METHOD tree->node_set_exp_image
    EXPORTING  node_key   = node_key
               exp_image  = exp_image
    EXCEPTIONS failed               = 1
               node_not_found       = 2
               cntl_system_error    = 3
               not_allowed_for_leaf = 4.
```

Parameters	Description
node_key	Name of the node for which you want to change the symbol. The parameter is defined with reference to type **`tv_nodekey`**.
exp_image	' ': Uses a default icon **`'@xy@'`**: Uses the SAP icon with number **`xy`** **`'BNONE'`**: No icon. As a result, the display position of the node is brought forwards.

You can address the icon using its name, for example, **`ICON_ANNOTATION`**. To do this, the statement **`INCLUDE <ICON>.`** must appear in your program.

node_set_hidden

Use this method to hide a particular node.

```
CALL METHOD tree->node_set_hidden
    EXPORTING  node_key   = node_key
            hidden     = hidden
    EXCEPTIONS failed              = 1
            node_not_found      = 2
            cntl_system_error    = 3.
```

Parameters	Description
node_key	Node that you want to hide The parameter is defined with reference to type **tv_nodekey**.
hidden	' ': Node is visible '**X**': Node, and all of its children, are invisible

node_set_is_folder

The **is_folder** attribute defines a node as a branch. This means that you can assign child nodes to it.

```
CALL METHOD tree->node_set_is_folder
   EXPORTING  node_key   = node_key
          is_folder  = is_folder
   EXCEPTIONS failed              = 1
          node_not_found      = 2
          cntl_system_error    = 3
          node_has_children    = 4.
```

Parameters	Description
node_key	Name of the node for which you want to set the **is_folder** attribute. The parameter is defined with reference to type **tv_nodekey**.
is_folder	'**X**': Sets the **is_folder** attribute for the node. ' ': Does not set the **is_folder** attribute for the node.

node_set_no_branch

This method controls whether to draw the hierarchy line to a node.

```
CALL METHOD tree->node_set_no_branch
   EXPORTING  node_key   = node_key
          no_branch  = no_branch
   EXCEPTIONS failed              = 1
          node_not_found      = 2
          cntl_system_error    = 3.
```

Parameters	Description
node_key	Node for which you want to change the hierarchy line. The parameter is defined with reference to type **tv_nodekey**.
branch	' ': Node with hierarchy line '**X**': Node without hierarchy line

node_set_n_image

Use this method to change the symbol for a leaf (node with no subordinate nodes) or an unexpanded branch (node with subordinate nodes).

```
CALL METHOD tree->node_set_n_image
    EXPORTING  node_key   = node_key
             n_image    = n_image
    EXCEPTIONS failed               = 1
             node_not_found      = 2
             cntl_system_error    = 3.
```

Parameters	**Description**
node_key	Node for which you want to change the symbol. The parameter is defined with reference to type **`tv_nodekey`**.
n_image	' ': Uses a default icon **`'@xy@'`**: Uses the SAP icon with number **`xy`** **`'BNONE'`**: No icon. As a result, the display position of the node is brought forwards.

You can address the icon using its name, for example, **`ICON_ANNOTATION`**. To do this, the statement **`INCLUDE <ICON>.`** must appear in your program.

node_set_style

Sets the style of a node.

```
CALL METHOD tree->node_set_style
    EXPORTING  node_key   = node_key
             style      = style
    EXCEPTIONS failed              = 1
             node_not_found      = 2
             cntl_system_error    = 3.
```

Parameters	Description
node_key	Node for which you want to change the style. The parameter is defined with reference to type **tv_nodekey**.
style	Style of the item. You can use one of the following styles: tree->style_default tree->style_inherited tree->style_intensified tree->style_inactive tree->style_intensified_critical tree->style_emphasized_negative tree->style_emphasized_positive tree->style_emphasized

remove_all_key_strokes

Use this method to reset all key registrations you made using add_key_stroke [Page 120].

```
CALL METHOD tree->remove_all_key_strokes
   EXCEPTIONS failed           = 1
          cntl_system_error = 2.
```

scroll

Use this method to scroll through the tree.

```
CALL METHOD tree->scroll
    EXPORTING scroll_command = scroll_command
    EXCEPTIONS failed              = 1
            illegal_scroll_command = 2
            cntl_system_error      = 3.
```

Parameters	Description
scroll_command	**tree->sroll_up_line** scrolls up one line. **tree->sroll_down_line** scrolls down one line. **tree->sroll_up_line** scrolls up one page. **tree->sroll_down_page** scrolls down one page. **tree->scroll_home** scrolls to the beginning of the tree. **tree->scroll_end** scrolls to the end of the tree.

select_nodes

Use this method to select a set of nodes in the tree.

You can only use it if you set multiple selection **(NODE_SELECTION_MODE = TREEV_SELECT_NODES)** when you created the tree control.

```
CALL METHOD tree->select_nodes
    EXPORTING  node_key_table = node_key_table
    EXCEPTIONS failed                    = 1
           cntl_system_error          = 2
           error_in_node_key_table      = 3
           dp_error                   = 4
           multiple_node_selection_only = 5.
```

Parameters	Description
node_key_table	Node table containing the nodes you want to select. Declare the node table with reference to the type **treev_nks**.

set_default_drop

Use this method to set a drag and drop behavior for the drop event on the control background.

```
CALL METHOD tree->set_default_drop
   EXPORTING  drag_drop     = drag_drop
   EXCEPTIONS failed             = 1
          cntl_system_error     = 2
          invalid_drag_drop_obj = 3.
```

Parameters	Description
drag_drop	Drag and drop behavior [Page 106] to be assigned to the background of the control.

set_folder_show_exp_image

Use this method to set the folder symbol you want to use for an open folder.

```
CALL METHOD tree->set_folder_show_exp_image
    EXPORTING  folder_show_exp_image = folder_show_exp_image
    EXCEPTIONS failed           = 1
            cntl_system_error = 2.
```

Parameters	Description
folder_show_exp_image	'**X**': Open folders always display the symbol specified in the **exp_image** field of the node. ' ': Only the last folder to be opened displays the folder symbol entered in the **exp_image** field of the node. The other folders display the folder symbol specified in the **n_image** field for the node.

set_has_3d_frame

Use this method to specify whether the SAP Tree should appear with a three-dimensional border.

```
CALL METHOD tree->set_has_3d_frame
    EXPORTING  has_3d_frame = has_3d_frame
    EXCEPTIONS failed          = 1
            cntl_system_error = 2.
```

Parameters	Description
has_3d_frame	'**x**': The SAP Tree is displayed in a 3D frame ' ': The control appears "flat" on the screen.

set_screen_update

Use this method to control whether the tree is refreshed. Use the **UPDATE** parameter to determine whether changes to the tree control should be visible immediately.

Use this method if the tree control is redrawn too many times in quick succession due to a series of changes to the data.

Using it will improve the performance of your program. Call it at the beginning of the PAI event using **UPDATE = ' '** and then again at the end of the PBO event using **UPDATE = 'X'**.

```
CALL METHOD tree->set_screen_update
    EXPORTING  UPDATE = UPDATE
    exceptions failed          = 1
             cntl_system_error = 2.
```

Parameters	Description
UPDATE	**'X'**: All changes are visible immediately **' '**: The changes are not visible immediately They become visible when you call the method again with **UPDATE = 'X'**.

set_selected_node

Use this method to select a particular node within the tree.

You can only use it if you set multiple selection **(NODE_SELECTION_MODE = tree->node_sel_mode_single**) when you created the tree control.

```
CALL METHOD tree->set_selected_node
    EXPORTING  node_key   = node_key
    EXCEPTIONS failed                    = 1
           single_node_selection_only = 2
           node_not_found            = 3
           cntl_system_error         = 4.
```

Parameters	Description
node_key	Node that you want to select. The parameter is defined with reference to type **tv_nodekey**.

set_top_node

When you use this method the system scrolls the tree so that the specified node appears at the top of the display if possible.

```
CALL METHOD tree->set_top_node
    EXPORTING  node_key   = node_key
    EXCEPTIONS failed                    = 1
               node_not_found           = 2
               cntl_system_error        = 3.
```

Parameters	Description
node_key	Node that you want to appear at the top of the display. The parameter is defined with reference to type **`tv_nodekey`**.

unselect_all

Use this method to deselect any selected nodes in the tree.

```
CALL METHOD tree->unselect_all
   EXCEPTIONS failed                  = 1
          cntl_system_error        = 2.
```

unselect_nodes

Use this method to deselect a list of selected nodes in the tree.

```
CALL METHOD tree->unselect_nodes
    EXPORTING  node_key_table = node_key_table
    EXCEPTIONS failed                    = 1
            cntl_system_error            = 2
            error_in_node_key_table      = 3
            dp_error                     = 4
            multiple_node_selection_only = 5.
```

Parameters	Description
node_key_table	Table containing the nodes that you want to deselect. Declare the node table with reference to the type `treev_nks`.

set_ctx_menu_select_event_appl

Use this method to set whether the event that occurs after the user has chosen an entry from a context menu should be an application event or a system event. The default is a system event.

```
CALL METHOD tree->set_ctx_menu_select_event_appl
    EXPORTING  appl_event = appl_event.
```

Methods of Class CL_ITEM_TREE_CONTROL

You can use these methods with both the column tree and the list tree.

add_nodes_and_items

Use this method to add nodes and items to the tree. To do this, you need an internal table containing the nodes and items you want to insert.

The node table is processed at the frontend in the order in which you filled it. Consequently, if you insert a node, you must ensure that its parent node has already occurred in the node table.

Tree control proxy objects (in this case, **`list_tree`** or **`column_tree`**) does not itself contain any data. Instead, you use it to transfer data to and from the SAP Tree at the frontend. You must maintain the tree data structure in your application.

```
CALL METHOD tree->add_nodes_and_items

   EXPORTING  node_table              = node_table
          item_table             = item_table
          item_table_structure_name = item_table_structure_name
   EXCEPTIONS failed                   = 1
          cntl_system_error           = 2
          error_in_tables            = 3
          dp_error                  = 4
          table_structure_name_not_found = 5.
```

Parameters	Description
node_table	Name of the nodes internal table. Create the table with reference to **`treev_ntab`**.
item_table	Name of the internal table containing the nodes for insertion. Define the table with reference to a structure of your own.
item_table_structure_name	Name of the structure used to create the internal table for the entries

Filling the Node Table

The node table structure consists of the following fields. You must fill the structure for each node.

node_key	Name of the node you want to define. The name must be a unique key within the tree. The node must not already exist.
relatkey	Name of the related node. This must already be defined in the tree. It must therefore come above the current node in the node table. If the value is initial, the node is inserted as the root node.

relatship	Relationship between **node_key** and **relatkey**: **tree->relat_first_child** (**node_key** is inserted as the first child node of node **relatkey**) **tree->relat_first_child** (**node_key** is inserted as the first child node of node **relatkey**) **tree->relat_prev_sibling** (**node_key** is inserted before **relatkey** at the same hierarchy level) **tree->relat_prev_sibling** (**node_key** is inserted after **relatkey** at the same hierarchy level) **tree->relat_first_sibling** (**node_key** is inserted before all other nodes at the same hierarchy level as **relatkey**). **tree->relat_last_sibling** (**node_key** is inserted after all other nodes at the same hierarchy level as **relatkey**).
hidden	initial: Node is displayed **'X'**: Node is not displayed
disabled	initial: Node can be selected **'X'**: Node cannot be selected
isfolder	initial: Node has no subordinate nodes **'X'**: Node has subordinate nodes
n_image	Specifies an icon for an unexpanded branch or a leaf: ' ': Uses a default icon **'@xy@'**: Uses the icon with number **xy** **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.
exp_image	Specifies an icon for an expanded branch or a leaf: ' ': Uses a default icon **'@xy@'**: Uses the icon with number **xy** **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.
last_hitem	Name of the last item that you want to appear under the hierarchy heading. (Can only be used with the list tree.)
no_branch	initial: Draws a hierarchy line to the node **'X'**: Suppresses the hierarchy line to the node
no_branch	initial: Draws a connecting line to the node. **'X'**: Suppresses the connecting line to the node.

expander	initial: Node has no '+' sign for expansion. This setting is only valid for nodes with no child nodes. **'X'**: Node must be a branch (**ISFOLDER = 'X'**) and has a '+' sign for expansion. If the user expands a branch that has no children, the event **EXPAND_NO_CHILDREN** is triggered.
dragdropid	Use this field for a handle to the drag and drop behavior of the node (see also Defining Drag and Drop Events in the SAP Tree [Page 111]).

Creating the ABAP Dictionary Structure for the Item Table

When you create the ABAP Dictionary structure <**my_item**>, you must include the structure treev_item and add an extra text field with the name **Text**. Define the text field using a text type.

Filling the Item Table

node_key	Name of the node to which the item should belong.
item_name	Name of the column in which you want to display this item. For the list tree, you can use numbers ≥ 1. In the column tree, specify the previously-defined column.
class	**tree->item_class_text**: Item is text **tree->item_class_checkbox**: Item as checkbox **tree->item_class_button**: Item is a pushbutton **tree->item_class_link**: Item is a link
font	Font: **tree->item_font_default**: corresponds to **tree->item_font_prop** in the tree structure but to **tree->item_font_fixed** in the list tree. **tree->item_font_fixed:** GUI fixed font **tree->item_font_prop**: GUI proportional font.
disabled	Deactivates an item
editable	Sets whether an item can be edited
hidden	Sets the visibility of an item
alignment	Alignment of an item (only in list tree)
t_image	Icon for the item
chosen	Checkbox selected
togg_right	You can only use togg_right for items with the class **TREE->ITEM_CLASS_CHECKBOX**. If **TOGG_RIGHT** is initial, the checkbox appears to the left of the text. Otherwise, it appears to the right of the text.

style	Style of the item. You can use one of the following styles: tree->style_default tree->style_inherited tree->style_intensified tree->style_inactive tree->style_intensified_critical tree->style_emphasized_negative tree->style_emphasized_positive tree->style_emphasized
length	Visible length of the item (only in list tree)
length_pix	Length in pixels (only in list tree)
ignoreimag	Can only be used in the list tree. initial: **LENGTH** contains the length of the item text. The width of any checkbox or icon is added to the width of the text. **'X': LENGTH** contains the width of the whole item. In this case, icons take up space that would otherwise be occupied by text.
usebgcolor	Can only be used in the list tree. **'X'**: The item has a background color that is slightly different to that of the tree control. initial: The background color is the same.
text	Text for the item.

If the user changes the font, the change does not take effect until the next instantiation of the SAP Tree.

You can address the icon using its name, for example, **ICON_ANNOTATION**. To do this, the statement **INCLUDE <ICON>.** must appear in your program.

delete_all_items_of_nodes

Use this method to delete all items of a particular node from the tree.

```
ALL METHOD tree->DELETE_ALL_ITEMS_OF_NODES
   EXPORTING  node_key_table = node_key_table
   EXCEPTIONS failed                  = 1
           cntl_system_error       = 2
           error_in_node_key_table  = 3
           dp_error                = 4.
```

Parameters	Description
node_key_table	Node table Nodes whose items will be deleted.

delete_items

Use this method to delete the items from the tree that you pass to it in a table.

```
CALL METHOD tree->delete_items
   EXPORTING  item_key_table = item_key_table
   EXCEPTIONS failed                        = 1
          cntl_system_error          = 2
          error_in_item_key_table      = 3
          dp_error                      = 4.
```

Parameters	Description
item_key_table	Table of entries that you want to delete.

get_selected_item

This method returns a selected item and its node.

```
CALL METHOD tree->get_selected_item
   IMPORTING  node_key  = node_key
          item_name = item_name
   EXCEPTIONS failed            = 1
          cntl_system_error  = 2
          no_item_selection  = 3.
```

Parameters	Description
node_key	Node in the SAP Tree with a selected item. The parameter is defined with reference to type **`tv_nodekey`**.
item_name	Name of the selected item. The parameter is defined with reference to type **`tv_itmname`** .

item_set_chosen

Use this method to select or deselect a checkbox from within your program.

```
CALL METHOD tree->item_set_chosen
    EXPORTING  node_key  = node_key
            item_name = item_name
            chosen    = chosen
    EXCEPTIONS failed              = 1
            node_not_found      = 2
            item_not_found      = 3
            cntl_system_error   = 4
            chosen_not_supported = 5.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to select. The parameter is defined with reference to type **tv_nodekey**.
item_name	Name of the checkbox item that you want to select. The parameter is defined with reference to type **tv_itmname** .
chosen	'**X**': Item is set to selected. ' ': Item is set to deselected.

item_set_disabled

Use this method to deactivate an item.. This item cannot then be selected. Furthermore, no other actions, such as double-clicking, are possible.

```
CALL METHOD tree->item_set_disabled
    EXPORTING  node_key  = node_key
            item_name = item_name
            disabled  = disabled
    EXCEPTIONS failed              = 1
            node_not_found      = 2
            item_not_found      = 3
            cntl_system_error    = 4
            no_item_selection    = 5.
```

Parameters	Description
node_key	Node containing the item you want to deactivate. The parameter is defined with reference to type **`tv_nodekey`**.
item_name	Name of the item that you want to deactivate. The parameter is defined with reference to type **`tv_itmname`** .
disabled	'**`X`**': Item is inactive ' ': Item is active

item_set_editable

Use this method to control whether the user can change a checkbox.

When you initialize the control, the parameter `ITEM_SELECTION` must be set to `'X'`.

The item must be a checkbox (the field `class` in structure `TREEV_ITEM` must have the value `tree->item_class_checkbox`).

```
CALL METHOD tree->item_set_editable
    EXPORTING  node_key  = node_key
            item_name = item_name
            editable  = editable
    EXCEPTIONS failed                = 1
            node_not_found        = 2
            item_not_found        = 3
            cntl_system_error     = 4
            editable_not_supported = 5.
```

Parameters	Description
node_key	Node containing the item you want to make accept input. The parameter is defined with reference to type `tv_nodekey`.
item_name	Name of the item that you want to make accept input. The parameter is defined with reference to type `tv_itmname` .
editable	'`X`': Checkbox can be changed. ' ': Checkbox cannot be changed.

item_set_font

Use this method to change the font of the text of an item.

```
CALL METHOD tree->item_set_font
    EXPORTING  node_key  = node_key
            item_name = item_name
            font      = font
    EXCEPTIONS failed               = 1
            node_not_found        = 2
            item_not_found        = 3
            cntl_system_error      = 4.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to change. The parameter is defined with reference to type **tv_nodekey**.
item_name	Name of the item whose font you want to change. The parameter is defined with reference to type **tv_itmname** .
font	Font: **tree->item_font_default:** corresponds to **tree->item_font_prop** in the tree structure but to **tree->item_font_fixed** in the list tree. tree->item_font_fixed: GUI fixed font **tree->item_font_prop**: GUI proportional font.

item_set_hidden

Use this method to hide a particular item of a node.

```
CALL METHOD tree->item_set_hidden
   EXPORTING  node_key  = node_key
         item_name = item_name
         hidden    = hidden
   EXCEPTIONS failed             = 1
         node_not_found       = 2
         item_not_found       = 3
         cntl_system_error     = 4.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to hide. The parameter is defined with reference to type **`tv_nodekey`**.
item_name	Name of the item that you want to hide. The parameter is defined with reference to type **`tv_itmname`** .
hidden	' ': Item is visible '**`X`**': Entry is no longer visible

item_set_style

Use this method to set the style for the combination of background and foreground color for the item.

```
CALL METHOD tree->item_set_style
    EXPORTING  node_key  = node_key
              item_name = item_name
              style     = style
    EXCEPTIONS failed             = 1
              node_not_found     = 2
              item_not_found     = 3
              cntl_system_error  = 4.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to change. The parameter is defined with reference to type **tv_nodekey**.
item_name	Name of the item whose style you want to change. The parameter is defined with reference to type **tv_itmname** .
style	Style of the entry. You can use one of the following styles: tree->style_default tree->style_inherited tree->style_intensified tree->style_inactive tree->style_intensified_critical tree->style_emphasized_negative tree->style_emphasized_positive tree->style_emphasized

item_set_text

Use this method to set or change the text of an item.

```
CALL METHOD tree->item_set_text
    EXPORTING  node_key  = node_key
            item_name = item_name
            text      = text
    EXCEPTIONS failed               = 1
            node_not_found        = 2
            item_not_found        = 3
            cntl_system_error      = 4.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to change. The parameter is defined with reference to type **`tv_nodekey`**.
item_name	Name of the item whose text you want to change. The parameter is defined with reference to type **`tv_itmname`** .
text	Text to be assigned to the entry.

item_set_t_image

Use this method to set an icon for an item. This allows you to set an icon as an item with or without a text.

```
CALL METHOD tree->item_set_t_image
    EXPORTING  node_key  = node_key
           item_name = item_name
           t_image   = t_image
    EXCEPTIONS failed              = 1
           node_not_found        = 2
           item_not_found        = 3
           cntl_system_error      = 4.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to change. The parameter is defined with reference to type **`tv_nodekey`**.
item_name	Name of the item for which you want to set an icon. The parameter is defined with reference to type **`tv_itmname`** .
t_image	' ': No icon. **`'@xy@'`**: Uses the SAP icon with number **`xy`**

You can address the icon using its name, for example, **`ICON_ANNOTATION`**. To do this, the statement **`INCLUDE <ICON>.`** must appear in your program.

select_item

Use this method to select a particular item within the tree.

> ➡
>
> When you initialize the control, the parameter **ITEM_SELECTION** must be set to **'X'**.

```
CALL METHOD tree->select_item
    EXPORTING  node_key  = node_key
            item_name = item_name
    EXCEPTIONS failed                  = 1
            key_or_item_name_not_found = 2
            no_item_selection          = 3
            cntl_system_error          = 4.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to change. The parameter is defined with reference to type **tv_nodekey**.
item_name	Name of the item that you want to select. The parameter is defined with reference to type **tv_itmname** .

set_min_node_height

Use this method to set the minimum height of a node.

```
CALL METHOD tree->set_min_node_height
   EXPORTING  include_text     = include_text
          include_image    = include_image
          include_button   = include_button
          include_checkbox = include_checkbox
          include_link     = include_link
   EXCEPTIONS failed           = 1
          cntl_system_error  = 2.
```

Parameters	Description
include_text	If you set this flag, the node is at least as high as a text object.
include_image	If you set this flag, the node is at least as high as a picture object (folder or leaf symbol).
include_button	If you set this flag, the node is at least as high as a pushbutton.
include_checkbox	If you set this flag, the node is at least as high as a checkbox
include_link	If you set this flag, the node is at least as high as a link entry.

update_nodes_and_items

Use this method to change a set of node and item attributes.

```
CALL METHOD tree->UPDATE_NODES_AND_ITEMS
    exporting node_table              = node_table
            item_table             = item_table
            ITEM_TABLE_STRUCTURE_NAME = ITEM_TABLE_STRUCTURE_NAME
    EXCEPTIONS failed                        = 1
             cntl_system_error            = 2
             error_in_tables              = 3
             dp_error                    = 4
             TABLE_STRUCTURE_NAME_NOT_FOUND = 5.
```

Parameters	Description
node_table	Name of the nodes internal table. Compared to the normal node table, this table also contains the structure **treemunode**. Use this to determine the attributes you want to change. Create the table with reference to **treev_upno**.
item_table	Name of the internal table containing the nodes you want to change. Define the table with reference to a structure of your own.
item_table_structure_name	Name of the structure used to create the internal table for the entries

Filling the Node Table

The node table structure consists of the following fields. You must fill the structure for each node.

node_key	Name of the node you want to change. The name must exist in the tree.
hidden	initial: Node is displayed **'X'**: Node is not displayed
disabled	initial: Node can be selected **'X'**: Node cannot be selected
isfolder	initial: Node has no subordinate nodes Note that the node may not have subordinate nodes. **'X'**: Node has subordinate nodes
n_image	Specifies an icon for an unexpanded branch or a leaf: ' ': Uses a default icon **'@xy@'**: Uses the icon with number **xy** **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.

exp_image	Specifies an icon for an expanded branch or a leaf: ' ': Uses a default icon **'@xy@'**: Uses the icon with number **xy** **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.
style	Node style.
no_branch	initial: Draws a connecting line to the node. **'X'**: Suppresses the connecting line to the node.
expander	initial: Node has no '+' sign for expansion. **'X'**: Node must be a branch (**ISFOLDER = 'X'**) and has a '+' sign for expansion. If the user expands a branch that has no children, the event **EXPAND_NO_CHILDREN** is triggered.
u_all	Change all changeable attributes
u_hidden	Change the **hidden** attribute.
u_disabled	Change the **disabled** attribute.
u_isfolder	Change the **is_folder** attribute.
u_n_image	Change the **n_image** attribute.
u_exp_imag	Change the **exp_image** attribute.
u_style	Change the **style** attribute.
u_no_branch	Change the **no_branch** attribute.
u_expander	Change the **expander** attribute.

Suppose you want to change the **hidden** and **is_folder** attributes: You assign values to the **hidden** and **is_folder** fields. The flags **u_hidden** and **u_is_folder** are set, to select the fields **hidden** and **is_folder** for change.

If you choose the field **u_all**, all of the fields for which a "U flag" exists are selected for change.

Creating the ABAP Dictionary Structure for the Item Table

When you create the ABAP Dictionary structure <**my_u_item**>, you must include the structure **treev_uite** and add an extra text field with the name **Text**. Define the text field using a text type.

Filling the Item Table

node_key	Name of the node containing the item you want to check.

item_name	Name of the column in which you want to change the item. For the list tree, you can use numbers ≥ 1. In the column tree, specify the previously-defined column.
class	**tree=>item_class_text**: Item is a text **tree=>item_class_checkbox**: Item is a checkbox **tree=>item_class_button**: Item is a pushbutton **tree=>item_class_link**: Item is a link item
font	Font: **tree->item_font_default**: corresponds to **tree->item_font_prop** in the tree structure but to **tree->item_font_fixed** in the list tree. tree->item_font_fixed: GUI fixed font **tree->item_font_prop**: GUI proportional font.
disabled	Deactivates an entry
editable	Sets whether an entry can be edited
hidden	Sets the visibility of an entry
alignment	Sets the alignment of an entry (only in list structure)
t_image	Icon for the entry
chosen	Selects a checkbox
togg_right	You can only use togg_right for items with the class **TREE->ITEM_CLASS_CHECKBOX**. If **TOGG_RIGHT** is initial, the checkbox appears to the left of the text. Otherwise, it appears to the right of the text.
style	Style of the entry. You can use one of the following styles: tree->style_default tree->style_inherited tree->style_intensified tree->style_inactive tree->style_intensified_critical tree->style_emphasized_negative tree->style_emphasized_positive tree->style_emphasized
length	Visible length of the entry (only in list structure)
length_pix	Length in pixels (only in list structure)

ignoreimag	Can only be used in the list tree. initial: **LENGTH** contains the length of the item text. The width of any checkbox or icon is added to the width of the text. **'X': LENGTH** contains the width of the whole item. In this case, icons take up space that would otherwise be occupied by text.
usebgcolor	Can only be used in the list tree. **'X'**: The item has a background color that is slightly different to that of the tree control. initial: The background color is the same.
text	Text for the entry.
u_all	Changes all modifiable attributes
u_font	Changes the font
u_disabled	Changes the **disabled** attribute.
u_editable	Changes the **editable** attribute.
u_hidden	Changes the **hidden** attribute.
u_alignmen	Changes the **alignment** attribute.
u_t_image	Changes the **t_image** attribute.
u_chosen	Changes the **chosen** attribute.
u_style	Changes the **style** attribute.
u_text	Changes the **text** attribute.
u_length	Changes the **length** attribute.
u_length_p	Changes the **length_pix** attribute.

⇨

If the user changes the font, the change does not take effect until the next instantiation of the SAP Tree.

Methods of Class CL_GUI_SIMPLE_TREE

constructor

You use this method to instantiate the simple tree.

```
CREATE OBJECT simple_tree
    EXPORTING  lifetime           = lifetime
          parent            = parent
          shellstyle         = shellstyle
          node_selection_mode = node_selection_mode
          hide_selection      = hide_selection
    EXCEPTIONS lifetime_error             =  1
          cntl_system_error         = 2
          create_error              = 3
          failed                    = 4
          illegal_node_selection_mode = 5.
```

Parameters	Description
lifetime	Lifetime management [Ext.] parameter. The following values are permitted: **simple_tree->lifetime_imode**: The control remains alive for the duration of the internal session (that is, until the session is ended by one of the following statements: **leave program. leave to transaction. set screen 0, leave screen.**). After this, the finalize [Page 482] method is called. **simple_tree->lifetime_dynpro**: The control remains alive for the lifetime of the screen instance, that is, for as long as the screen remains in the stack. After this, the free [Page 480] method is called. Using this mode automatically regulates the visibility of the control. Controls are only displayed when the screen on which they were created is active. When other screens are active, the controls are hidden. **simple_tree->lifetime_default**: If you create the control in a container, it inherits the lifetime of the container. If you do not create the control in a container (for example, because it is a container itself), the lifetime is set to **simple_tree->lifetime_imode**.
parent	Container in which the SAP Tree can be displayed (**see also** SAP Container [Ext.]).
node_selection_mode	**simple_tree->node_sel_mode_single**: Only single selection allowed. **simple_tree->node_sel_mode_multiple**: Multiple selection allowed.
hide_selection	Hides a selection

add_nodes

Use this method to add nodes to an existing tree. You pass the list of new entries using an internal table. The internal table must be defined with reference to an ABAP Dictionary structure of your own.

The node table is processed at the frontend in the order in which you filled it. Consequently, if you insert a node, you must ensure that its parent node has already occurred in the node table.

⚠

A tree control proxy object (in this case, **simple_tree**) does not itself contain any data. Instead, you use it to transfer data to and from the SAP Tree at the frontend. You must maintain the tree data structure in your application.

```
CALL METHOD simple_tree->add_nodes
    EXPORTING  table_structure_name = table_structure_name
           node_table          = node_table
    EXCEPTIONS error_in_node_table          = 1
           failed                     = 2
           dp_error                     = 3
           table_structure_name_not_found = 4.
```

Parameters	Description
node_table_structure_name	Name of the structure used to create the internal table for the entries
node_table	Internal table containing the nodes you want to create

Creating the ABAP Dictionary Structure

When you create the ABAP Dictionary structure <**my_node**>, you must include the structure **treev_node** and add an extra text field with the name **Text**. Define the text field using a text type.

Filling the Node Table

The node table structure consists of the following fields. You must fill the structure for each node.

node_key	Name of the node you want to define. The name must be a unique key within the tree. The node must not already exist.
relatkey	Name of the related node. This must already be defined in the tree. It must therefore come above the current node in the node table.

relatship	Relationship between **node_key** and **relatkey**: **tree->relat_first_child** (**node_key** is first child node of node **relatkey**) **tree->relat_first_child** (**node_key** is inserted as the first child node of node **relatkey**) **tree->relat_prev_sibling** (**node_key** is inserted before **relatkey** at the same hierarchy level) **tree->relat_prev_sibling** (**node_key** is inserted after **relatkey** at the same hierarchy level) **tree->relat_first_sibling** (**node_key** is inserted before all other nodes at the same hierarchy level as **relatkey**). **tree->relat_last_sibling** (**node_key** is inserted after all other nodes at the same hierarchy level as **relatkey**).
hidden	initial: Node is displayed **'X'**: Node is not displayed
disabled	initial: Node can be selected **'X'**: Node cannot be selected
isfolder	initial: Node has no subordinate nodes **'X'**: Node has subordinate nodes
n_image	Specifies an icon for an unexpanded branch or a leaf: ' ': Uses a default icon **'@xy@'**: Uses the icon with number xy **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.
exp_image	Specifies an icon for an expanded branch or a leaf: ' ': Uses a default icon **'@xy@'**: Uses the icon with number xy **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.

style	Style of the entry. You can use one of the following styles: tree->style_default tree->style_inherited tree->style_intensified tree->style_inactive tree->style_intensified_critical tree->style_emphasized_negative tree->style_emphasized_positive tree->style_emphasized
no_branch	initial: Draws a connecting line to the node. `'X'`: Does not draw a connecting line to the node.
expander	initial: Node has no ‘+’ sign for expansion. `'X'`: Node must be a branch (**`ISFOLDER = ‘X’`**) and has a ‘+’ sign for expansion. If the user expands a branch that has no children, the event **EXPAND_NO_CHILDREN** is triggered.
dragdropid	Use this field for a handle to the drag and drop behavior of the node (see also Defining Drag and Drop Events in the SAP Tree [Page 111]).
text	This field is only used in the simple tree. It is part of your node structure definition in the ABAP Dictionary. When you use a simple tree, use this field to pass the text you want to display.

You can address the icon using its name, for example, **`ICON_ANNOTATION`**. To do this, the statement **`INCLUDE <ICON>.`** must appear in your program.

node_set_text

Use this method to change the text of a node.

```
CALL METHOD simple_tree->node_set_text
    EXPORTING  node_key  = node_key
           text      = text
    EXCEPTIONS failed             = 1
           node_not_found      = 2
             cntl_system_error = 3.
```

Parameters	Description
node_key	Name of the node for which you want to change the text. The parameter is defined with reference to type **`tv_nodekey`**.
text	Text to be assigned to the node.

update_nodes

Use this method to changes the attributes of a set of nodes (text, relationship to other nodes, display options). You need a special node table, which you create with reference to an ABAP Dictionary structure that you have to define yourself.

To change a particular attribute of a node, you must set the corresponding flag.

```
CALL METHOD simple_tree->update_nodes
   EXPORTING  node_table_structure_name = node_table_structure_name
          node_table               = node_table
   EXCEPTIONS failed                       = 1
          cntl_system_error            = 2
          error_in_node_table           = 3
          dp_error                     = 4
          table_structure_name_not_found = 5.
```

Parameters	Description
node_table_structure_name	Name of the structure used to create the internal table for the changed nodes.
node_table	Internal table containing the changed nodes.

Creating the ABAP Dictionary Structure

To create the ABAP Dictionary structure <**my_unode**>, you must include the structure **treev_unod** and create two additional fields - one called **Text** (a text field), the other called **U_TEXT** with type **AS4FLAG**. Define the text field using a text type.

Filling the Node Table

The node table structure consists of the following fields. You must fill the structure for each node.

node_key	Name of the node you want to define. The name must be a unique key within the tree.
relatkey	Name of the related node. This must already be defined in the tree. It must therefore come above the current node in the node table.

relatship	Relationship between **`node_key`** and **`relatkey`**: **`simple_tree->relat_first_child`** (**`node_key`** is first child node of node **`relatkey`**) **`simple_tree->relat_last_child`** (**`node_key`** is the last child node of node **`relatkey`**) **`simple_tree->relat_prev_sibling`** (**`node_key`** is inserted before **`relatkey`** at the same hierarchy level) **`simple_tree->relat_prev_sibling`** (**`node_key`** is inserted after **`relatkey`** at the same hierarchy level) **`simple_tree->relat_first_sibling`** (**`node_key`** is inserted before all other nodes at the same hierarchy level as **`relatkey`**). **`simple_tree->relat_last_sibling`** (**`node_key`** is inserted after all other nodes at the same hierarchy level as **`relatkey`**).
hidden	initial: Node is displayed `'X'`: Node is not displayed
disabled	initial: Node can be selected `'X'`: Node cannot be selected
isfolder	initial: Node has no subordinate nodes `'X'`: Node has subordinate nodes
n_image	Specifies an icon for an unexpanded branch or a leaf: ' ': Uses a default icon **'@xy@'**: Uses the icon with number **`xy`** **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.
exp_image	Specifies an icon for an expanded branch or a leaf: ' ': Uses a default icon **'@xy@'**: Uses the icon with number **`xy`** **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.
style	Node style.
no_branch	initial: Draws a connecting line to the node. `'X'`: Suppresses the connecting line to the node.
expander	initial: Node has no '+' sign for expansion. `'X'`: Node must be a branch (`ISFOLDER = 'X'`) and has a '+' sign for expansion. If the user expands a branch that has no children, the event **EXPAND_NO_CHILDREN** is triggered.

text	This field is only used in the simple tree. It is part of your node structure definition in the ABAP Dictionary. When you use a simple tree, use this field to pass the text you want to display.
u_all	Change all changeable attributes
u_hidden	Change the **hidden** attribute.
u_disabled	Change the **disabled** attribute.
u_isfolder	Change the **is_folder** attribute.
u_n_image	Change the **n_image** attribute.
u_exp_imag	Change the **exp_image** attribute.
u_style	Change the **style** attribute.
u_no_branch	Change the **no_branch** attribute.
u_expander	Change the **expander** attribute.
u_text	Change the node text.

Suppose you want to change the **hidden** and **is_folder** attributes: You assign values to the **hidden** and **is_folder** fields. The flags **u_hidden** and **u_is_folder** are set, to select the fields **hidden** and **is_folder** for change.

If you choose the field **u_all**, all of the fields for which a "U flag" exists are selected for change.

You can address the icon using its name, for example, **ICON_ANNOTATION**. To do this, the statement **INCLUDE <ICON>.** must appear in your program.

Methods of Class CL_GUI_LIST_TREE

constructor

You use this method to instantiate the list tree.

```
CREATE OBJECT list_tree
    EXPORTING  lifetime            = lifetime
          parent              = parent
          shellstyle          = shellstyle
          node_selection_mode = node_selection_mode
          hide_selection      = hide_selection
          item_selection      = item_selection
          with_headers        = with_headers
          hierarchy_header    = hierarchy_header
          list_header         = list_header
    EXCEPTIONS lifetime_error            =  1
          cntl_system_error         = 2
          create_error              = 3
          illegal_node_selection_mode = 4
          failed                    = 5.
```

Parameters	Description
lifetime	Lifetime management [Ext.] parameter. The following values are permitted: **list_tree->lifetime_imode**: The control remains alive for the duration of the internal session (that is, until the session is ended by one of the following statements: **leave program. leave to transaction. set screen 0, leave screen.**). After this, the finalize [Page 482] method is called. **list_tree->lifetime_dynpro**: The control remains alive for the lifetime of the screen instance, that is, for as long as the screen remains in the stack. After this, the free [Page 480] method is called. Using this mode automatically regulates the visibility of the control. Controls are only displayed when the screen on which they were created is active. When other screens are active, the controls are hidden. **list_tree->lifetime_default**: If you create the control in a container, it inherits the lifetime of the container. If you do not create the control in a container (for example, because it is a container itself), the lifetime is set to **simple_tree->lifetime_imode**.
parent	Container in which the SAP Tree can be displayed (**see also** SAP Container [Ext.]).
node_selection_mode	**list_tree->node_sel_mode_single**: Only single selection allowed. **list_tree->node_sel_mode_multiple**: Multiple selection allowed.

constructor

hide_selection	Hides a selection
item_selection	Flags whether individual entries should be selectable. If you set this attribute, the node can only be selected using its icon. If the attribute is not set, the entire node can be selected as a single unit.
with_headers	Flags whether headers are used.
hierarchy_header	Structure with the definition of the first header.
list_header	Structure with the definition of the following header.

node_set_last_hierarchy_item

Use this method to determine the item in a node that should be the last to appear under the hierarchy heading. All subsequent items then appear under the list heading.

```
CALL METHOD list_tree->node_set_last_hierarchy_item
    EXPORTING node_key          = node_key
           last_hierarchy_item = last_hierarchy_item
      EXCEPTIONS failed             = 1
             node_not_found     = 2
             cntl_system_error  = 3
             tree_has_no_headers = 4.
```

Parameters	Description
node_key	Node you want to change. The parameter is defined with reference to type **`tv_nodekey`**.
last_hierarchy_item	Name of the last item that you want to appear under the hierarchy heading. The parameter is defined with reference to type **`tv_itmname`** .

hierarchy_header_set_t_image

Use this method to define an icon for the hierarchy heading.

```
CALL METHOD list_tree->hierarchy_header_set_t_image
   EXPORTING t_image = t_image
    EXCEPTIONS tree_has_no_headers = 1
          failed            = 2
          cntl_system_error   = 3.
```

Parameters	Description
t_image	' ': No icon. **'@xy@'**: Uses the SAP icon with number **xy**

You can address the icon using its name, for example, **ICON_ANNOTATION**. To do this, the statement **INCLUDE <ICON>.** must appear in your program.

list_header_set_t_image

Use this method to define an icon for the list heading.

```
CALL METHOD list_tree->LIST_HEADER_SET_T_IMAGE
   EXPORTING t_image = t_image
    EXCEPTIONS tree_has_no_headers = 1
          failed              = 2
          cntl_system_error   = 3.
```

Parameters	Description
t_image	' ': No icon. '@xy@': Uses the SAP icon with number xy 'BNONE': No icon. As a result, the display position of the node is brought forwards.

You can address the icon using its name, for example, `ICON_ANNOTATION`. To do this, the statement `INCLUDE <ICON>.` must appear in your program.

hierarchy_header_set_text

You use this method to change the text of the hierarchy heading:

```
CALL METHOD list_tree->hierarchy_header_set_text
    EXPORTING text = text
    EXCEPTIONS tree_has_no_headers = 1
            failed            = 2
            cntl_system_error   = 3.
```

Parameters	Description
text	Hierarchy heading text The parameter is defined with reference to type **`tv_heading`** .

list_header_set_text

You use this method to change the text of the list heading:

```
CALL METHOD list_tree->list_header_set_text
    EXPORTING text = text
    EXCEPTIONS tree_has_no_headers = 1
            failed            = 2
            cntl_system_error   = 3.
```

Parameters	Description
text	List heading text The parameter is defined with reference to type **`tv_heading`** .

hierarchy_header_set_width

Use this method to set the width of the hierarchy heading.

```
CALL METHOD list_tree->hierarchy_header_set_width
    EXPORTING width      = width
            width_pix = width_pix
    EXCEPTIONS tree_has_no_headers = 1
             failed              = 2
             cntl_system_error   = 3.
```

Parameters	Description
width	Width of the heading
width_pix	'X': The length is interpreted in pixels. ' ': The length is interpreted in characters.

hiearchy_header_get_width

This method returns the current width of the hierarchy heading in pixels.

```
CALL METHOD list_tree->hierarchy_header_get_width
    IMPORTING width     = width
            width_pix = width_pix
    EXCEPTIONS failed             = 1
             cntl_system_error   = 2
             tree_has_no_headers = 3.
```

Parameters	Description
width	Width of the heading
width_pix	'X': Width in pixels

hierarchy_header_adjust_width

Use this method to adjust the width of the headings so that the entire contents of the items below them are visible.

```
CALL METHOD list_tree->hierarchy_header_adjust_width
    EXCEPTIONS failed            = 1
           cntl_system_error   = 2
           tree_has_no_headers = 3.
```

item_set_alignment

Use this method to set the width of an item.

```
CALL METHOD list_tree->item_set_alignment
    EXPORTING node_key  = node_key
            item_name = item_name
            alignment = alignment
    EXCEPTIONS failed             = 1
             node_not_found     = 2
             item_not_found     = 3
             cntl_system_error  = 4.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to change. The parameter is defined with reference to type **`tv_nodekey`**.
item_name	Name of the item for which you want to adjust the alignment. The parameter is defined with reference to type **`tv_itmname`** .
alignment	**`list_tree->align_left`**: left justified **`list_tree->align_right`**: right-justified **`list_tree->align_center`**: centered **`list_tree->align_auto`**: automatic alignment

item_set_length

Use this method to change the displayed length of a particular item.

```
CALL METHOD list_tree->item_set_length
    EXPORTING node_key   = node_key
           item_name  = item_name
           length      = length
           length_pix = length_pix
    EXCEPTIONS failed              = 1
           node_not_found      = 2
           item_not_found      = 3
           cntl_system_error   = 4.
```

Parameters	Description
node_key	Node in the SAP Tree containing the item that you want to change. The parameter is defined with reference to type **tv_nodekey**.
item_name	Name of the item for which you want to adjust the alignment. The parameter is defined with reference to type **tv_itmname** .
length	Display length for the item.
length_pix	'**X**': The length is interpreted in pixels. ' ': The length is interpreted in characters.

hierarchy_header_set_tooltip

Use this method to set a tooltip for a hierarchy heading. The tooltip is displayed whenever the mouse pointer is positioned over the hierarchy heading.

```
CALL METHOD list_tree->hierarchy_header_set_tooltip
   EXPORTING tooltip = tooltip
   EXCEPTIONS tree_has_no_headers = 1
          failed          = 2
          cntl_system_error   = 3.
```

Parameters	Description
tooltip	Text

list_header_set_tooltip

Use this method to set a tooltip for a list heading. The tooltip is displayed whenever the mouse pointer is positioned over the list heading.

```
CALL METHOD list_tree->list_header_set_tooltip
    EXPORTING tooltip = tooltip
    EXCEPTIONS tree_has_no_headers = 1
            failed              = 2
            cntl_system_error   = 3.
```

Parameters	Description
tooltip	Text

Methods of Class CL_GUI_COLUMN_TREE

constructor

You use this method to instantiate the column tree.

```
CREATE OBJECT column_tree
    EXPORTING  lifetime            = lifetime
          parent             = parent
          shellstyle          = shellstyle
          node_selection_mode   = node_selection_mode
          hide_selection       = hide_selection
          item_selection       = item_selection
          hierarchy_column_name = hierarchy_column_name
          hierarchy_header      = hierarchy_header
    EXCEPTIONS lifetime_error            =  1
          cntl_system_error          = 2
          create_error              = 3
          illegal_node_selection_mode = 4
          failed                    = 5
          illegal_column_name        = 6.
```

Parameters	Description
lifetime	Lifetime management [Ext.] parameter. The following values are permitted: **`column_tree->lifetime_imode`**: The control remains alive for the duration of the internal session (that is, until the session is ended by one of the following statements: **`leave program. leave to transaction. set screen 0, leave screen.`**). After this, the finalize [Page 482] method is called. **`column_tree->lifetime_dynpro`**: The control remains alive for the lifetime of the screen instance, that is, for as long as the screen remains in the stack. After this, the free [Page 480] method is called. Using this mode automatically regulates the visibility of the control. Controls are only displayed when the screen on which they were created is active. When other screens are active, the controls are hidden. **`column_tree->lifetime_default`**: If you create the control in a container, it inherits the lifetime of the container. If you do not create the control in a container (for example, because it is a container itself), the lifetime is set to **`simple_tree->lifetime_imode`**.
parent	Container in which the SAP Tree can be displayed (**see also** SAP Container [Ext.]).
node_selection_mode	**`column_tree->node_sel_mode_single`**: Only single selection allowed. **`column_tree->node_sel_mode_multiple`**: Multiple selection allowed.

hide_selection	Hides a selection
item_selection	Flags whether individual entries should be selectable. If you set this attribute, the node can only be selected using its icon. If the attribute is not set, the entire node can be selected as a single unit.
hierarchy_column_name	Name of the column heading
hierarchy_header	Structure with the definition of the first header.

add_column

Use this method to add a new column to the tree. The column has its own heading - it is not inserted under the hierarchy heading.

```
CALL METHOD column_tree->add_column
   EXPORTING  name         = name
          hidden        = hidden
          disabled       = disabled
          alignment      = alignment
          width          = width
          width_pix      = width_pix
          header_image   = header_image
          header_text    = header_text
          header_tooltip = header_tooltip
    EXCEPTIONS column_exists              = 1
           illegal_column_name         = 2
           too_many_columns            = 3
           illegal_alignment           = 4
           different_column_types      = 5
           cntl_system_error           = 6
           failed                      = 7
           predecessor_column_not_found = 8.
```

Parameters	Description
name	Technical name of the column
hidden	' ': Column is visible '**X**': Column is not visible
disabled	'**X**': Column can be selected ' ': Column cannot be selected
alignment	**`column_tree->align_left`**: left justified **`column_tree->align_right`**: right-justified **`column_tree->align_center`**: centered
width	Width of the heading
width_pix	'**X**': The width is interpreted in pixels. ' ': The width is interpreted in characters.
header_image	' ': No icon. `'@xy@'`: Uses the SAP icon with number **`xy`**
header_text	Hierarchy heading text The parameter is defined with reference to type **`tv_heading`** .
header_tooltip	Text that is displayed when the mouse pointer is positioned on the heading.

You can address the icon using its name, for example, **`ICON_ANNOTATION`**. To do this, the statement **`INCLUDE <ICON>.`** must appear in your program.

add_hierarchy_column

Use this method to insert a new column. It is inserted below the hierarchy heading.

```
CALL METHOD column_tree->add_hierarchy_column
   EXPORTING  name           = name
          hidden        = hidden
          disabled       = disabled
   EXCEPTIONS column_exists              =  1
          illegal_column_name         = 2
          too_many_columns            = 3
          cntl_system_error           = 4
          failed                 = 5.
```

Parameters	Description
name	Technical name of the column
hidden	' ': Column is visible 'X': Column is not visible
disabled	'X': Column cannot be selected ' ': Column can be selected

You should only insert one text column below a hierarchy heading. Further columns should contain icons, checkboxes, or other classes of item.

adjust_column_width

Use this method to adjust the width of the headings so that the entire contents of the items below them are visible. You can either adjust all columns or specify a range (column n to column m) to be adjusted.

```
CALL METHOD column_tree->adjust_column_width
   EXPORTING  start_column = start_column
          end_column   = end_column
          all_columns  = all_columns
   EXCEPTIONS start_column_not_found    = 1
          end_column_not_found      = 2
          start_column_in_hierarchy = 3
          end_column_in_hierarchy   = 4
          start_column_empty        = 5
          cntl_system_error         = 6
          failed                    = 7.
```

Parameters	Description
start_column	Technical name of the first column outside the column heading that you want to adjust.
end_column	Technical name of the last column outside the column heading that you want to adjust.
all_columns	'**X**': All columns, including the hierarchy header, are adjusted.

column_get_width

This method returns the current width of the specified column. The width is returned in pixels.

```
CALL METHOD column_tree->column_get_width
    EXPORTING column_name = column_name
    IMPORTING width       = width
    EXCEPTIONS failed               = 1
            column_not_found     = 2
            hierarchy_column     = 3
            cntl_system_error    = 4.
```

Parameters	Description
column_name	Technical name of the column
width	Width of the heading

column_set_disabled

Use this method to deactivate a column. The column cannot then be selected. Furthermore, no other actions, such as double-clicking, are possible.

```
CALL METHOD column_tree->column_set_disabled
    EXPORTING column_name = column_name
            disabled    = disabled
    EXCEPTIONS failed               = 1
             column_not_found     = 2
             cntl_system_error    = 3.
```

Parameters	Description
column_name	Technical name of the column
disabled	'X': Column is inactive ' ': Column is active

column_set_heading_image

Use this method to define an icon for the column heading.

```
CALL METHOD column_tree->column_set_heading_image
    EXPORTING column_name = column_name
            image       = image
    EXCEPTIONS failed              = 1
             column_not_found     = 2
             hierarchy_column     = 3
             cntl_system_error    = 4.
```

Parameters	Description
column_name	Technical name of the column
image	' ': No icon. **'@xy@'**: Uses the SAP icon with number **xy** **'BNONE'**: No icon. As a result, the display position of the node is brought forwards.

You can address the icon using its name, for example, **ICON_ANNOTATION**. To do this, the statement **INCLUDE <ICON>.** must appear in your program.

column_set_heading_text

Use this method to change the text of the column heading:

```
CALL METHOD column_tree->column_set_heading_text
    EXPORTING column_name = column_name
           text        = text
    EXCEPTIONS failed               = 1
            column_not_found     = 2
            hierarchy_column     = 3
            cntl_system_error    = 4.
```

Parameters	Description
column_name	Technical name of the column
text	Hierarchy heading text The parameter is defined with reference to type **`tv_heading`** .

column_set_heading_tooltip

Use this method to set a tooltip for the heading. The tooltip is displayed whenever the mouse pointer is positioned over the heading.

```
CALL METHOD column_tree->column_set_heading_tooltip
    EXPORTING column_name = column_name
            tooltip     = tooltip
    EXCEPTIONS failed               = 1
             column_not_found     = 2
             hierarchy_column     = 3
             cntl_system_error    = 4.
```

Parameters	Description
column_name	Technical name of the column
tooltip	Text

column_set_hidden

Use this method to hide a particular column.

```
CALL METHOD column_tree->column_set_hidden
    EXPORTING column_name = column_name
            hidden      = hidden
    EXCEPTIONS failed              = 1
            column_not_found      = 2
            cntl_system_error     = 3.
```

Parameters	Description
column_name	Technical name of the column
hidden	' ': Column is visible 'X': Column is not visible

column_set_width

Use this method to set the width of the column heading.

```
CALL METHOD column_tree->column_set_width
    EXPORTING column_name = column_name
            width       = width
            width_pix   = width_pix
    EXCEPTIONS failed              = 1
             column_not_found     = 2
             hierarchy_column     = 3
             cntl_system_error    = 4.
```

Parameters	Description
column_name	Technical name of the column
width	Width of the heading
width_pix	'X': The width is interpreted in pixels. ' ': The width is interpreted in characters.

delete_column

Use this method to delete a column.

```
CALL METHOD column_tree->delete_column
    EXPORTING column_name = column_name
    EXCEPTIONS failed              = 1
            column_not_found      = 2
            cntl_system_error     = 3.
```

Parameters	Description
column_name	Technical name of the column

get_column_order

This method returns the sequence of the columns. This is useful if you want to find out if the user moved any columns.

```
CALL METHOD column_tree->get_column_order
    CHANGING   columns = columns
    EXCEPTIONS cntl_system_error = 1
           dp_error         = 2
           failed           = 3.
```

Parameters	Description
columns	Internal table, containing the columns in the order in which they appear in the tree. The parameter is defined with reference to type **`treev_cona`** .

hierarchy_header_adjust_width

Use this method to adjust the width of the hierarchy heading so that the entire contents of the columns are visible.

```
CALL METHOD column_tree->hierarchy_header_adjust_width
    EXCEPTIONS failed              = 1
               cntl_system_error   = 2.
```

hierarchy_header_get_width

This method returns the current width of the hierarchy heading. The width is returned in pixels.

```
CALL METHOD column_tree->hierarchy_header_get_width
    IMPORTING width     = width
            width_pix = width_pix
    EXCEPTIONS failed               = 1
             cntl_system_error     = 2.
```

Parameters	Description
width	Width of the heading
width_pix	'X': Width in pixels

hierarchy_header_set_text

You use this method to change the text of the hierarchy heading:

```
CALL METHOD column_tree->hierarchy_header_set_text
    EXPORTING text = text
    EXCEPTIONS failed              = 1
            cntl_system_error     = 2.
```

Parameters	Description
text	Hierarchy heading text The parameter is defined with reference to type **`tv_heading`** .

hierarchy_header_set_tooltip

Use this method to set a tooltip for a hierarchy heading. The tooltip is displayed whenever the mouse pointer is positioned over the heading.

```
CALL METHOD column_tree->hierarchy_header_set_tooltip
     EXPORTING  tooltip = tooltip
     EXCEPTIONS failed                = 1
                cntl_system_error     = 2.
```

Parameters	Description
tooltip	Text

hierarchy_header_set_t_image

Use this method to define an icon for the hierarchy heading.

```
CALL METHOD column_tree->hierarchy_header_set_t_image
    EXPORTING  t_image = t_image
    EXCEPTIONS failed              = 1
            cntl_system_error     = 2.
```

Parameters	Description
t_image	' ': No icon. **'@xy@'**: Uses the SAP icon with number **xy**

You can address the icon using its name, for example, **ICON_ANNOTATION**. To do this, the statement **INCLUDE <ICON>.** must appear in your program.

hierarchy_header_set_width

Use this method to set the width of the hierarchy heading.

```
CALL METHOD column_tree->hierarchy_header_set_width
    EXPORTING  width     = width
            width_pix = width_pix
    EXCEPTIONS failed              = 1
            cntl_system_error     = 2.
```

Parameters	Description
width	Width of the heading
width_pix	**'X'**: The length is interpreted in pixels. ' ': The length is interpreted in characters.

insert_column

Use this method to insert a new column with a heading after an existing column with heading in the tree.

```
CALL METHOD column_tree->insert_column
   EXPORTING  name              = name
          predecessor_column = predecessor_column
          hidden            = hidden
          disabled          = disabled
          alignment         = alignment
          width             = width
          width_pix         = width_pix
          header_image      = header_image
          header_text       = header_text
          header_tooltip    = header_tooltip
   EXCEPTIONS column_exists             =  1
          illegal_column_name       = 2
          too_many_columns          = 3
          illegal_alignment         = 4
          different_column_types    = 5
          cntl_system_error         = 6
          failed                    = 7
          predecessor_column_not_found = 8.
```

Parameters	Description
name	Technical name of the column
predecessor_column	Technical name of the preceding column The preceding column cannot be a hierarchy column. initial: The column is inserted as the first column after the hierarchy columns.
hidden	' ': Column is visible 'X': Column is not visible
disabled	'X': Column cannot be selected ' ': Column can be selected
alignment	**column_tree->align_left**: left justified **column_tree->align_right**: right-justified **column_tree->align_center**: centered
width	Width of the heading
width_pix	'X': The width is interpreted in pixels. ' ': The width is interpreted in characters.
header_image	' ': No icon. '@xy@': Uses the SAP icon with number **xy**

header_text	Hierarchy heading text The parameter is defined with reference to type **`tv_heading`** .
header_tooltip	Text that is displayed when the mouse pointer is positioned on the heading.

You can address the icon using its name, for example, **`ICON_ANNOTATION`**. To do this, the statement **`INCLUDE <ICON>.`** must appear in your program.

insert_hierarchy_column

Use this method to insert a new column after an existing column. It is inserted below the hierarchy heading.

```
CALL METHOD column_tree->insert_hierarchy_column
   EXPORTING  name              = name
          predecessor_column = predecessor_column
          hidden            = hidden
          disabled          = disabled
   EXCEPTIONS column_exists              =  1
          illegal_column_name        = 2
          too_many_columns           = 3
          different_column_types     = 4
          cntl_system_error          = 5
          failed                     = 6
          predecessor_column_not_found = 7.
```

Parameters	Description
name	Technical name of the column
predecessor_column	Technical name of the preceding column
hidden	' ': Column is visible 'X': Column is not visible
disabled	'X': Column cannot be selected ' ': Column can be selected

set_column_order

Use this method to set a new sequence for the columns.

```
CALL METHOD column_tree->set_column_order
    EXPORTING  columns = columns
    EXCEPTIONS cntl_system_error = 1
           dp_error          = 2
           failed            = 3
           column_not_found  = 4
           hierarchy_column  = 5
           wrong_column_set  = 6.
```

Parameters	Description
columns	Internal table containing the defined columns in their new sequence. The parameter is defined with reference to type **treev_cona** .

update_column

Use this method to change the attributes of a column.

```
CALL METHOD column_tree->update_column
    EXPORTING  name             = name
            hidden           = hidden
            disabled         = disabled
            alignment        = alignment
            header_image     = header_image
            header_text      = header_text
            header_tooltip   = header_tooltip
            update_width     = update_width
            width            = width
            width_pix        = width_pix
    EXCEPTIONS illegal_alignment  = 1
            cntl_system_error  = 2
            failed           = 3
            hierarchy_column   = 4
            column_not_found   = 5.
```

Parameters	Description
name	Technical name of the column
hidden	' ': Column is visible '**X**': Column is not visible
disabled	'**X**': Column cannot be selected ' ': Column can be selected
alignment	**list_tree->align_left**: left justified **list_tree->align_right**: right-justified **list_tree->align_center**: centered
header_image	' ': No icon. '**@xy@**': Uses the SAP icon with number **xy**
header_text	Hierarchy heading text The parameter is defined with reference to type **tv_heading** .
header_tooltip	Text that is displayed when the mouse pointer is positioned on the heading.
update_width	'X': Change the width of the column to the value in the parameter **width**. ' ': Column width remains unchanged.
width	Width of the heading
width_pix	'**X**': The length is interpreted in pixels. ' ': The length is interpreted in characters.

You can address the icon using its name, for example, **`ICON_ANNOTATION`**. To do this, the statement **`INCLUDE <ICON>.`** must appear in your program.

SAP Tree Model

Purpose

The SAP Tree Model has been introduced to complement the SAP Tree Control. Unlike the SAP Tree Control, which only displayed data without actually administering it, the SAP Tree Model holds all of the data that is to be displayed on the application server. Instead of communicating directly with the tree at the frontend, the application program has only to communicate with the tree model. The tree model also ensures optimal performance - an important aspect of tree control programming that was previously left to the programmer.

Implementation Considerations

Use the SAP Tree Model whenever you want to display data in a hierarchical tree format.

Features

Like the SAP Tree, the SAP Tree Model has three variants (see Overview of SAP Tree Model Classes [Page 232]):

- Simple tree
- List tree
- Column tree

The SAP Tree Model also contains features that make it more comfortable to use than the normal SAP Tree:

- Automatic synchronization between the tree model on the application server and the tree control at the frontend
- Automatic flush handling
- Search and print functions
- Checks on the validity of node data **before** it is sent to the frontend (reduces the risk of runtime error CNTL_ERROR)
- Automatic control of how much data is sent to the frontend and when
- Option to display the tree in tabular form
- A freely-usable object reference to which you can assign an application-related object.

Constraints

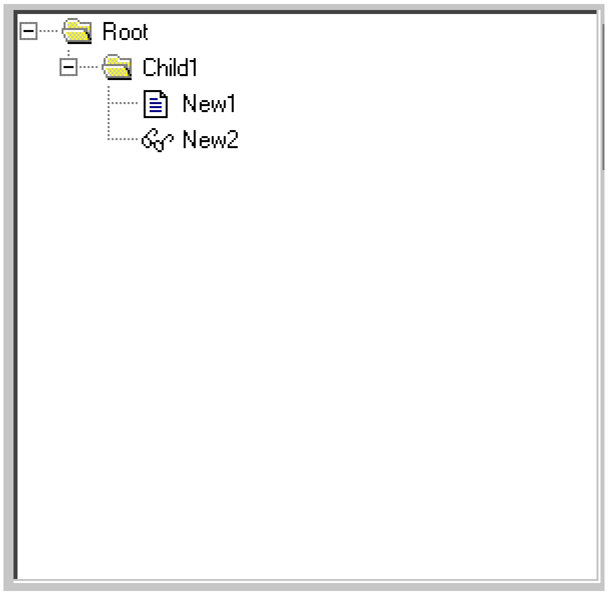
Certain features of the SAP Tree Model do not work in the SAPGUI for HTML environment. For further information, refer to The Tree Controls in SAPGUI for HTML [Page 14].

Overview of SAP Tree Model Classes

Simple Tree Model

The class **CL_SIMPLE_TREE_MODEL** is the ABAP Objects wrapper for the simple tree.

Example program: **SAPSIMPLE_TREE_MODEL_DEMO**:



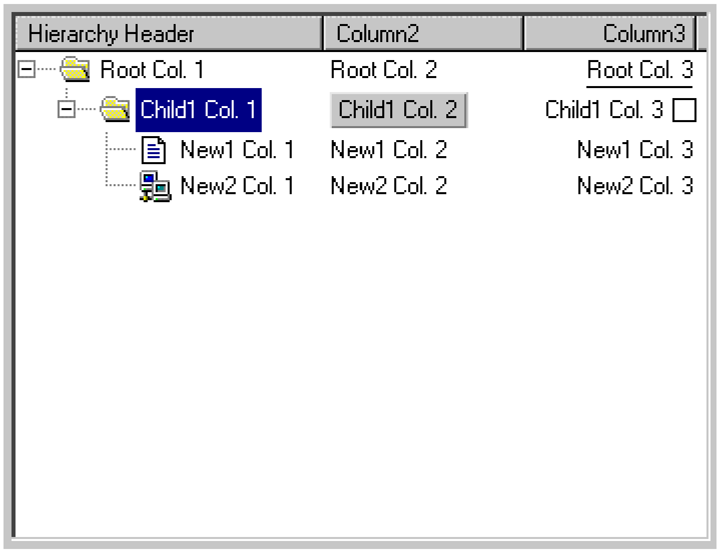
Attributes

- A node consists of a folder or leaf symbol and a text.
- You cannot use checkboxes or additional icons.
- You can only have one text for each node.
- There is no heading.

Column Tree Model

The class **CL_COLUMN_TREE_MODEL** is the ABAP Objects wrapper for the column tree model.

Example program: **SAPCOLUMN_TREE_MODEL_DEMO**:



Hierarchy Header	Column2	Column3
Root Col. 1	Root Col. 2	Root Col. 3
Child1 Col. 1	Child1 Col. 2	Child1 Col. 3 □
New1 Col. 1	New1 Col. 2	New1 Col. 3
New2 Col. 1	New2 Col. 2	New2 Col. 3

Attributes

- A node consists of a folder or leaf symbol and a range of items.
- The entries of a node are arranged in columns.

  In the example, the tree has three columns with the logical names 'Column1', 'Column2', and 'Column3'. The topmost node has an entry in each of these columns:

  'Root Col. 1' in column 'Column1'

  'Root Col. 2' in column 'Column2'

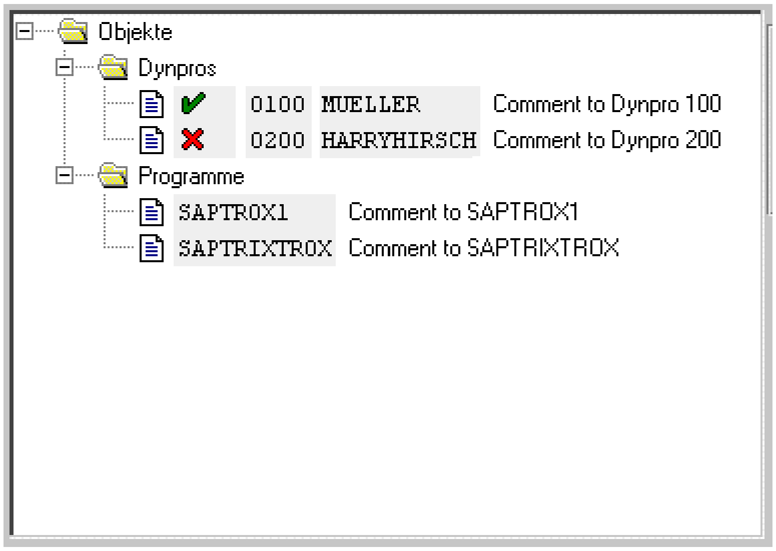
  'Root Col. 3' in column 'Column3'

- A column tree can contain two kinds of columns:
  - Columns in the hierarchy area: These columns are below the hierarchy heading. The hierarchy heading is the first heading from the left in the tree (in the example, 'Hierarchy Header'). There is normally only one column in the hierarchy area. In the example, it is the column with the name 'Column1', containing the entries 'Root Col.1', 'Child1 Col. 1' and so on.
  - Columns outside the hierarchy area: These columns have their own heading. The example contains two columns outside the hierarchy area, with the headings 'Column2' and 'Column3'.
- Columns can have the following kinds of entries:
  - Text: Text, with optional icon
  - Checkbox: Checkbox with optional icon and text.
  - Pushbutton: Pushbutton with text and icon.
  - Link: Like text, but additionally, an event is triggered when the user clicks the link.

List Tree Model

The class **CL_LIST_TREE_MODEL** is the ABAP Objects wrapper for the column tree.

Example program: **SAPLIST_TREE_MODEL_DEMO**:



```
⊟ Objekte
   ⊟ Dynpros
        ✓   0100 MUELLER     Comment to Dynpro 100
        ✗   0200 HARRYHIRSCH Comment to Dynpro 200
   ⊟ Programme
        SAPTROX1    Comment to SAPTROX1
        SAPTRIXTROX Comment to SAPTRIXTROX
```

Attributes

- A node consists of a folder or leaf symbol and entries.
- The entries are displayed from left to right.

  Structure of the first three nodes in the example:

  The topmost node has a single entry ("objects"). Proportional font is set for this entry. Additionally, the "automatic width" is set. This means that the width of the entry is adjusted to fit the contents (in this case, the string "objects").

  The second node from the top has the same construction as the first: An entry with the text "Screens".

  The third node from the top has four entries:

  A tick icon, four characters wide.

  0100, not in proportional font, four characters wide.

  MUELLER, not in proportional font, 11 characters wide.

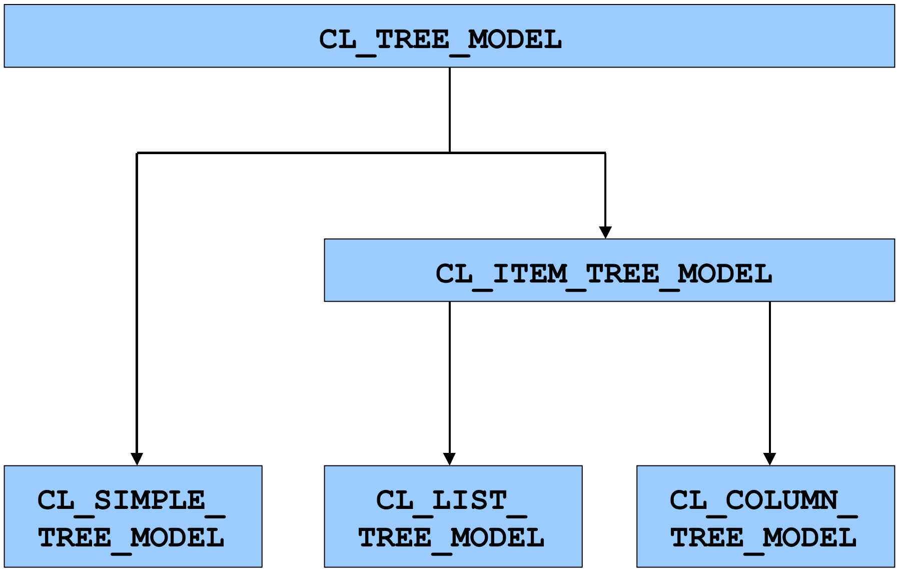
  Comment for screen 100, proportional font, automatic width.

- Using non-proportional fonts and a fixed display width allows you to display data in tabular format, as in the example.
- Columns can have the following kinds of entries:
  - Text: Text, with optional icon
  - Checkbox: Checkbox with optional icon and text.

  - Pushbutton: Pushbutton with text and icon.
  - Link: Like text, but additionally, an event is triggered when the user clicks the link.
- There is a hierarchy heading and a list heading, under which all entries can be grouped.

The Inheritance Hierarchy

The classes used in the SAP Tree Model form the following inheritance hierarchy:



CL_TREE_MODEL contains methods that are used by all three kinds of Tree Models. Additionally, **CL_LIST_TREE_MODEL** and **CL_COLUMN_TREE_MODEL** share certain methods, which are defined in their superclass **CL_ITEM_TREE_MODEL**.

Finding Errors

The majority of errors in control programming occur when you synchronize the automation queue [Ext.]. Synchronization occurs either explicitly, using the method CL_GUI_CFW=>FLUSH [Page 474], or implicitly after the last PBO module has finished.

If the error occurs in an explicit synchronization, the method **CL_GUI_CFW=>FLUSH** triggers the exception **CNTL_ERROR**. If the error occurs in an implicit synchronization, a short dump occurs. You can avoid the short dump by handling special events of the Control Framework.

The exception **CNTL_ERROR** only indicates that an unspecified method call to a control at the frontend was unsuccessful. You then need to find out which control at the frontend has triggered the exception and why. You can do this using the Debugger:

5. Run the program again in the Debugger.
6. Go into the settings in the Debugger and select the option *Automation Controller: Always process requests synchronously*.
   When you set this option, the automation queue is synchronized after each method call.
7. Step through the individual method calls. Note that **SY-SUBRC** is only set after the method that triggers the exception if you handle the exceptions in your application program. Otherwise, another short dump occurs.
8. Identify the error in the method call.

💡

If an error occurs, you should first run the example programs for the corresponding control wrapper. If an error also occurs in these programs, the problem is due to your local SAPgui installation.

⚠

Once **CNTL_ERROR** has been triggered, you should no longer work with the controls. Remember above all that method calls that come after the error in the automation queue will not be processed.

If the error occurred in the first automation queue synchronization, the automation controller may no longer be active. This results in all subsequent control calls ending with a **CNTL_ERROR**.

Important Notes

The exceptions of the SAP Tree Model do not set messages.

You must never ignore exceptions of the SAP Tree Model methods or flush calls. If an error occurs, the automation queue processing is terminated. This affects all of the controls in the same internal session. Once an error has occurred, the internal session affected may no longer work with controls.

The SAP Tree Model is not suitable for displaying non-hierarchical lists, since all root nodes must always be transferred to the frontend. Consequently, long lists can cause performance problems.

Example Programs

The development class **SEU_TREE_MODEL** contains the following example programs that demonstrate how to program the three different kinds of tree models:

Program name	Demonstrates
SAPSIMPLE_TREE_MODEL_DEMO	Simple tree model
SAP_SIMPLE_TREEM_DRAG_DROP_DEMO	Drag and drop with the simple tree model
SAPCOLUMN_TREE_MODEL_DEMO	Column tree model
SAP_LIST_TREE_MODEL_DEMO	List tree model

Using Controls in a WAN

When you use controls in your programs, you place an extra load on the communication channel between the frontend and backend. In a LAN, and particularly in a WAN environment, this can be a critical factor.

The problem is alleviated somewhat by buffering mechanisms (see also Automation Queue [Ext.]). Use these points as a guideline to using controls in a WAN.

The documentation for the individual controls also contains more specific notes about using that control in a WAN.

Using CL_GUI_CFW=>FLUSH

The method CL_GUI_CFW=>FLUSH [Page 474] synchronizes the automation queue and the ABAP variables in it. Calling it often generates a synchronous RFC call from the application server to the frontend. To optimize the performance of your application, you should call this method as little as possible.

It is often a good idea to read all control attributes in a single automation queue (for example, at the beginning of the PAI) and retrieve them in a single synchronization. You should, in particular, do this when you read attributes that are not necessary in your event handlers or the PAI/PBO cycle.

You do not need to include a "safety flush" at the end of the PBO to ensure that all method calls are transported to the frontend. A flush at the end of the PBO is guaranteed. Consequently, you cannot construct an automation queue spread over several screens.

There is no guarantee that an automation queue will be sent when you call `CL_GUI_CFW=>FLUSH`. The queue recognizes whether it contains any return values. If this is not the case, it is not sent.

If you have a queue with no return values, and want to ensure that it is synchronized, you can use the Control Framework method CL_GUI_CFW=>UPDATE_VIEW [Page 477]. You should only use this method if you absolutely need to update the GUI. For example, you might have a long-running application in which you want to provide the user with regular updates on the status of an action.

After you have read the attributes of a control, the contents of the corresponding ABAP variables are not guaranteed until after the next flush. The contents of the ABAP variables remain undefined until this call occurs. In the future, there will be cases in which this flush is unnecessary. They will be recognized by the automation queue and the corresponding flush call will be ignored.

Creating Controls and Passing Data

Creating controls and passing data to them is normally a one-off procedure, which in comparison to using normal screen elements can be very runtime-intensive. You should therefore not use any unnecessary controls, or pass unnecessary data to the controls that you are using.

A typical example is a tabstrip control with several tab pages. If the pages contain controls, you should consider using application server scrolling instead of local scrolling, and not loading the controls until the corresponding page is activated by the user. The same applies to passing data to the controls on tab pages.

If you want to differentiate between LAN and WAN environments when you pass data to a control, you can use the function module **`SAPGUI_GET_WANFLAG`**. In some applications, you may need to pass different amounts of data or use a complete fallback in a WAN application. The environment affects, for example, the number of same-level nodes that you can transfer to a tree control without having to introduce artificial intermediate levels.

Unlike screen elements, controls only have to be created and filled with data once. From a performance point of view, this means that they become more profitable the longer they exist. In applications that are called repeatedly, and therefore initialized repeatedly, controls can have a negative effect on performance. In applications that use the same screen for a long time, on the other hand, you may find that using controls results in improved performance.

You can always use the performance tools [Ext.] to check the advantages and disadvantages in terms of network load that using a control brings.

Storing Documents, Picture, and Other Data

Release 4.6A sees the introduction of a frontend cache for accessing documents from the Business Document Service (BDS). You are strongly recommended to store desktop documents, images, and other data in the BDS and not in the R/3 database. Documents from the BDS can be cached at the frontend, and therefore only have to be loaded over the network once.

The Simple Tree Model

Definition

You create a simple tree model instance with reference to the class `cl_simple_tree_model`:

```
DATA simple_model TYPE REF TO cl_simple_tree_model.
```

This instance gives you access to the methods of the following classes:

- `cl_tree_model` (see Methods of Class CL_TREE_MODEL [Page 258])
- `cl_simple_tree_model` (see Methods of Class CL_SIMPLE_TREE_MODEL [Page 325])

Use

The program `sapsimple_tree_model_demo` demonstrates how to use the simple tree model.

For details of the attributes of the simple tree, refer to the Overview of SAP Tree Model Classes [Page 232].

Getting Started With the Simple Tree Model

Purpose

This section describes how to create, use, and destroy an instance of the SAP Simple Tree Model.

Prerequisites

The process described here is an extension of the general process for using controls [Page 90] that is specific to the simple tree model. It does not contain all of the steps required to produce a valid instance of the control.

Process Flow

The program extracts are examples that do not necessarily illustrate all of the features of the control. For precise information, consult the reference section of this documentation.

Create the Instance for the Backend Model

1. Define a reference variable for the simple tree model:

```
DATA simple_model TYPE REF TO cl_simple_tree_model.
```

2. Create an instance of the SAP Simple Tree Model:

```
CREATE OBJECT simple_model
  EXPORTING node_selection_mode         = node_selection_mode
            hide_selection              = hide_selection
  EXCEPTIONS illegal_node_selection_mode = 1.
```

Create the Corresponding Frontend Control

3. Call the method **create_tree_control** for the **simple_model** instance. This creates the frontend control in which the data from the simple tree model will be displayed.

```
CALL METHOD simple_model->create_tree_control
  EXPORTING  parent                      = container
  EXCEPTIONS lifetime_error              = 1
             cntl_system_error           = 2
             create_error                = 3
             failed                      = 4
             tree_control_already_created = 5
```

The parent parameter must contain the reference to a SAP Container that you have already created. For further information, refer to the SAP Container [Ext.] documentation.

Register the Events

4. Register the events [Page 99] of the simple tree model that you want to use. The control supports the following events:

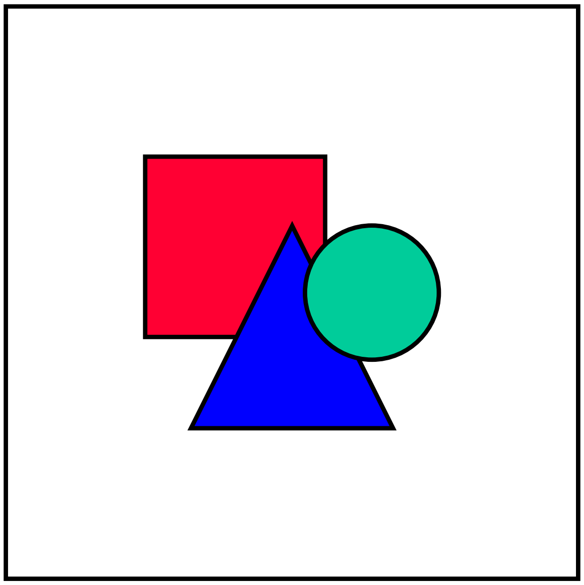
Event name	Meaning
NODE_DOUBLE_CLICK	User double-clicked a node
NODE_KEYPRESS	User pressed a certain key. The keys that trigger this event must be registered beforehand
EXPAND_NO_CHILDREN	User expanded a node that has no child nodes
SELECTION_CHANGED	Selected node has changed
NODE_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	User selected an entry from the context menu
DEFAULT_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an empty space in the control
DEFAULT_CONTEXT_MENU_SELECT	User selected an entry from the context menu

Fill the Simple Tree Model With Nodes

5. Add nodes to the simple tree model.

Fill a node table with the relevant node information, then pass it to the tree model using the add_nodes [Ext.] method:

```
CALL METHOD simple_model->add_nodes
      EXPORTING nodes_table = nodes_table
     EXCEPTIONS error_in_node_table = 1.
```



This step adds nodes at the backend. They are not transferred to the control on the screen until the end of the PBO event.

Work With the Tree Model

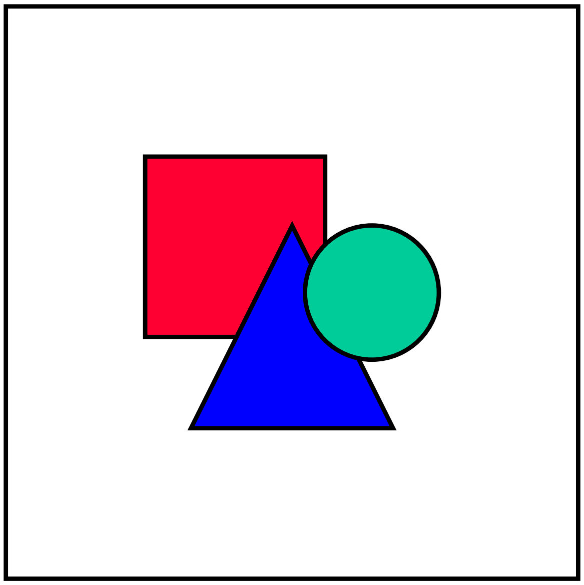
6. Find out any node attributes that you need.

7. Change any node attributes as required.

Destroy the Control

8. Destroy the control container at the frontend. This destroys the tree control instance contained within it as well.

```
CALL METHOD container->free.
```



If you are working with lifetime management, you do not need to worry about destroying the control at the frontend yourself. It is done automatically by the system instead.

9. Free the reference to the simple tree model. It will then be deleted by the garbage collector.

```
FREE simple_model.
```

Searching in the Simple Tree Model

Purpose

The Simple Tree Model, unlike the normal Simple Tree Control, allows you to search within the backend version of the tree using the following methods:

To search for	Use the following methods
Individual nodes	find [Page 338], find_first [Page 339], find_next [Page 341]
A set of nodes	find_all [Page 342], find_all_continue [Page 344]

Prerequisites

You must have created an instance of the Simple Tree Model.

Process Flow

1. Decide which search method you want to use and call the relevant method of `cl_simple_tree_model`:

Search method	Method of cl_simple_tree_model
Find individual nodes with user dialog	find [Page 338]
Find individual nodes without a user dialog	find_first [Page 339]
Find a set of nodes	find_all [Page 342]

2. After the search stops, query the value of the `result_type` parameter. This tells you whether the search text was found, not found, or if the search stopped because the system encountered a node with the attribute `EXPANDER = 'X'` and no child nodes.

3. If the search stopped because of the third case, you can now load the child nodes into the tree model using the add_node [Page 327] or add_nodes [Page 330] method, then restart the search:

Search method	Method used to restart the search
Individual nodes	find_next [Page 341]
A set of nodes	find_all_continue [Page 344]

If you are searching for individual nodes, you can also use `find_next` to go onto the next occurrence of the search string.

The Column Tree Model

Definition

You create a column tree model instance with reference to the class **cl_column_tree_model**:

```
DATA column_model TYPE REF TO cl_column_tree_model.
```

This instance gives you access to the methods of the following classes:

- **cl_tree_model** (see Methods of Class CL_TREE_MODEL [Page 258])
- **cl_item_tree_model** (see Methods of Class CL_ITEM_TREE_MODEL [Page 345])
- **cl_column_tree_model** (see Methods of Class CL_COLUMN_TREE_MODEL [Page 399])

Use

The program **sapcolumn_tree_model_demo** demonstrates how to use the column tree model.

For details of the attributes of the column tree, refer to the Overview of SAP Tree Model Classes [Page 232].

Getting Started With the Column Tree Model

This section lists the functions that are specific to the Column Tree Model.

Prerequisites

The process described here is an extension of the general process for using controls [Page 90] that is specific to the Column Tree Model. It does not contain all of the steps required to produce a valid instance of the control.

Process Flow

➡

The program extracts are examples that do not necessarily illustrate all of the features of the control. For precise information, refer to the reference section of this documentation.

Creating the Instance

28. Define a reference variable for the column tree model:

```
DATA column_model TYPE REF TO cl_column_tree_model.
```

29. Define a work area for the hierarchy heading by referring to the structure **TREEMHHDR**.

```
DATA hierarchy_header TYPE treemhhdr.
```

30. Fill the work area for the hierarchy heading. You can set the width (**width**), the text (**heading**), an icon (**image**) and a tool tip (**tooltip**). There are also methods that allow you to change these attributes later on.

```
hierarchy_header-heading = 'Title'.
hierarchy_header-width = 30.
```

31. Create an instance of the SAP Tree Model:

```
CREATE OBJECT column_model
     EXPORTING parent  = container
               node_selection_mode          = node_selection_mode
               hide_selection               = hide_selection
               item_selection               = item_selection
               hierarchy_column_name        = hierarchy_column_name
               hierarchy_header             = hierarchy_header
     EXCEPTIONS illegal_node_selection_mode = 1
                illegal_column_name         = 2.
```

Register the Events

32. Register the events [Page 101] of the Column Tree Model. The control supports the following events:

Event name	Description
NODE_DOUBLE_CLICK	User double-clicked a node
EXPAND_NO_CHILDREN	User expanded a node that has no children

SELECTION_CHANGED	Selected node has changed
NODE_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	User selected an entry from the context menu
DEFAULT_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an empty space in the control
DEFAULT_CONTEXT_MENU_SELECT	User selected an entry from the context menu
HEADER_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on the heading
HEADER_CONTEXT_MENU_SELECT	User selected an entry from the context menu
ITEM_KEYPRESS	User pressed a key while an entry was selected.
NODE_KEYPRESS	User pressed a key while an entry was selected.
HEADER_CLICK	User clicked a heading

If you set the parameter **`item_selection = 'X'`** when you created the instance, you can also react to the following events:

Event name	**Description**
BUTTON_CLICK	The user clicked an item with the class BUTTON
LINK_CLICK	The user clicked an item with the class LINK
CHECKBOX_CHANGE	The user clicked an item with the class CHECKBOX
ITEM_DOUBLE_CLICK	The user double-clicked an item
ITEM_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an item
ITEM_CONTEXT_MENU_SELECT	User selected an entry from the context menu for an item

Using the Column Tree

33. Add nodes to the tree. To do this, fill a node table (type **`TREEMCNOTA`**, line type TREEMCNODT [Page 466]), then pass it to the Column Tree Model using the method add_nodes [Page 404].

34. Add the items. To do this, fill an item table (type **`TREEMCITAC`**, line type **`TREEMCITEN`**), then pass it to the Column Tree Model using the method add_items [Page 406].

    ➡

    Remember that it is possible to update nodes and items at any time when you are working with the Column Tree Model. For further information, refer to update_nodes [Page 405] or update_items [Page 407].

8. Create the tree control instance that will display the data. Up until now, you have been working with the tree model on the application server. However, this cannot, of itself, display the data, so you now need to create the frontend tree. To do this, you must create a SAP

Container Control, then pass a reference to this container to the create_tree_control [Page 259] method:

```
CALL METHOD column_model->create_tree_control
          EXPORTING parent = container.
```

Loading Items on Demand

Use

In a very large List Tree Model or Column Tree Model, it may make sense not to load all of the items when you create the tree. Instead, you can load items "on demand", that is, when the user actually displays the node to which the items belong.

Prerequisites

- You must already have instantiated the List Tree Model or Column Tree Model
- You must have a class in your application that implements one of the following interfaces:
  - If you are using the List Tree Model, interface **IF_LIST_TREE_MODEL_ITEM_PROV**.
  - If you are using the Column Tree Model, interface **IF_COLUMN_TREE_MODEL_ITEM_PROV**.

Procedure

1. When you add new nodes to the tree model, set the flag **ITEMSINCOM** to **'X'**. This informs the tree model that you want to load the items for that node on demand.
2. In your application class, implement the method **LOAD_ITEMS** of the relevant interface (see the *Prerequisites* section above) so that it fills the internal table **item_table** with the attributes of the items you want to load.
3. Depending on which version of the tree model you are using, call one of the following interfaces and pass to it the instance of your application class that will provide the item information:

Tree Model version you are using	Method you should call
List Tree Model	set_item_provider [Page 397] of **cl_list_tree_model**
Column Tree Model	set_item_provider [Page 449] of **cl_column_tree_model**

Result

When the user displays a node for which the items have not yet been loaded, and for which you set the **ITEMSINCOM** attribute to **'X'**, it calls the **LOAD_ITEMS** method of the object you specified in the **SET_ITEM_PROVIDER** method. This loads the items into the tree model instance, after which, the system resets the **ITEMSINCOM** attribute to its initial value.

The List Tree Model

Definition

You create a list tree model instance with reference to the class **`cl_list_tree_model`**:

```
DATA list_model TYPE REF TO cl_list_tree_model.
```

This instance gives you access to the methods of the following classes:

- **`cl_tree_model`** (see Methods of Class CL_TREE_MODEL [Page 258])
- **`cl_item_tree_model`** (see Methods of Class CL_ITEM_TREE_MODEL [Page 345])
- **`cl_list_tree_model`** (see Methods of Class CL_LIST_TREE_MODEL [Page 369])

Use

The program **`saplist_tree_model_demo`** demonstrates how to use the list tree model.

For details of the attributes of the list tree, refer to the Overview of SAP Tree Model Classes [Page 232].

Getting Started With the List Tree Model

This section lists the functions that are specific to the List Tree Model.

Prerequisites

The process described here is an extension of the general process for using controls [Page 90] that is specific to the list tree. It does not contain all of the steps required to produce a valid instance of the control.

Process Flow

The program extracts are examples that do not necessarily illustrate all of the features of the control. For precise information, refer to the reference section of this documentation.

Creating the Instance

35. Define a reference variable for the List Tree Model:

```
DATA list_model TYPE REF TO cl_list_tree_model.
```

36. If you want to create a heading for the tree, you must create a work area for the hierarchy heading with reference to the structure **TREEMHHDR** and one for the list heading with reference to the structure **TREEMLHDR**:

```
DATA hierarchy_header TYPE treemhhdr.
DATA list_header type treemlhdr.
```

37. Fill the work area for the hierarchy heading. You can set the width (**width**), the text (**heading**), an icon (**t_image**) and a tool tip (**tooltip**). There are also methods that allow you to change these attributes later on.

```
hierarchy_header-heading = 'Title'.
hierarchy_header-width = 30.
```

38. Fill the work area for the list heading. You can set the text (**heading**), an icon (**t_image**) and a tool tip (**tooltip**).

```
list_header-heading = 'List heading'.
```

39. [Page 189]Create an instance of the SAP List Tree Model:

```
CREATE OBJECT list_model
       EXPORTING  node_selection_mode         = node_selection_mode
                  hide_selection              = hide_selection
                  item_selection              = item_selection
                  with_headers                = with_headers
                  hierarchy_header            = hierarchy_header
                  list_header                 = list_header
       EXCEPTIONS illegal_node_selection_mode = 1
```

Register the Events

40. Register the events for the List Tree Model. The control supports the following events:

Event name	Description
NODE_DOUBLE_CLICK	User double-clicked a node
EXPAND_NO_CHILDREN	User expanded a node that has no children
SELECTION_CHANGED	Selected node has changed
NODE_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on a node
NODE_CONTEXT_MENU_SELECT	User selected an entry from the context menu
DEFAULT_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an empty space in the control
DEFAULT_CONTEXT_MENU_SELECT	User selected an entry from the context menu
HEADER_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on the heading
HEADER_CONTEXT_MENU_SELECT	User selected an entry from the context menu
ITEM_KEYPRESS	User pressed a key while an entry was selected.
NODE_KEYPRESS	User pressed a key while an entry was selected.
HEADER_CLICK	User clicked a heading

If you set the parameter **`item_selection = 'X'`** when you created the instance, you can also react to the following events:

Event name	Description
BUTTON_CLICK	The user clicked an item with the class BUTTON
LINK_CLICK	The user clicked an item with the class LINK
CHECKBOX_CHANGE	The user clicked an item with the class CHECKBOX
ITEM_DOUBLE_CLICK	The user double-clicked an item
ITEM_CONTEXT_MENU_REQUEST	User requested a context menu with the cursor positioned on an item
ITEM_CONTEXT_MENU_SELECT	User selected an entry from the context menu for an item

Using the List Tree

7. Create a node table (type TREEMLNOTA, line type TREEMLNODT [Page 458]), fill it with the nodes you want to add to the List Tree Model, and then use the add_nodes [Page 375] method to pass the table to the tree model instance:

```
CALL METHOD list_model->add_nodes
        EXPORTING node_table = node_table.
```

9. Create an item table (type TREEMLITAC, line type TREEMLITEN), fill it with the items you want to add to the List Tree Mode, and then use the add_items method to pass the table to the tree model instance:

```
      CALL METHOD list_model->add_items
                EXPORTING item_table = item_table.
```

10. Create the tree control instance that will display the data. Up until now, you have been working with the tree model on the application server. However, this cannot, of itself, display the data, so you now need to create the frontend tree. To do this, you must create a SAP Container Control, then pass a reference to this container to the create_tree_control [Page 259] method:

```
CALL METHOD column_model->create_tree_control
          EXPORTING parent = container.
```

Loading Items on Demand

Use

In a very large List Tree Model or Column Tree Model, it may make sense not to load all of the items when you create the tree. Instead, you can load items "on demand", that is, when the user actually displays the node to which the items belong.

Prerequisites

- You must already have instantiated the List Tree Model or Column Tree Model
- You must have a class in your application that implements one of the following interfaces:
  - If you are using the List Tree Model, interface **`IF_LIST_TREE_MODEL_ITEM_PROV`**.
  - If you are using the Column Tree Model, interface **`IF_COLUMN_TREE_MODEL_ITEM_PROV`**.

Procedure

4. When you add new nodes to the tree model, set the flag **`ITEMSINCOM`** to **`'X'`**. This informs the tree model that you want to load the items for that node on demand.
5. In your application class, implement the method **`LOAD_ITEMS`** of the relevant interface (see the *Prerequisites* section above) so that it fills the internal table **`item_table`** with the attributes of the items you want to load.
6. Depending on which version of the tree model you are using, call one of the following interfaces and pass to it the instance of your application class that will provide the item information:

Tree Model version you are using	Method you should call
List Tree Model	set_item_provider [Page 397] of **`cl_list_tree_model`**
Column Tree Model	set_item_provider [Page 449] of **`cl_column_tree_model`**

Result

When the user displays a node for which the items have not yet been loaded, and for which you set the **`ITEMSINCOM`** attribute to **`'X'`**, it calls the **`LOAD_ITEMS`** method of the object you specified in the **`SET_ITEM_PROVIDER`** method. This loads the items into the tree model instance, after which, the system resets the **`ITEMSINCOM`** attribute to its initial value.

Processing Events in the Tree Model

Purpose

Certain user actions on the tree control in the SAP Tree Model cause it to trigger events. You can catch and handle the events in your ABAP program.

Process Flow

To react to an event from an SAP Tree Model instance, you must

- Define and implement a method (usually in a local class) as a handler for the event
- Register the event with the Control Framwork
- Register the event handler using the **SET HANDLER** statement.

For a full description of how to process control events, refer to Registering and Processing Events [Page 99].

Methods of Class CL_TREE_MODEL

create_tree_control

When you create a tree model instance, it cannot display the tree until you have called this method to create the frontend tree control.

```
CALL METHOD model->create_tree_control
          EXPORTING lifetime   = lifetime
                    parent     = parent
                    shellstyle = shellstyle
          IMPORTING control    = control.
```

Parameter and Type	Opt.	Description
lifetime **TYPE I**	X	The lifetime of the control. If you leave this parameter blank, the control inherits the lifetime of its container. Other possible values: • **cl_gui_control=>lifetime_imode**: The control remains alive for the duration of the internal session (that is, until the **leave program**, **leave to transaction**, or **set screen 0. leave screen** statements occur. • **cl_gui_control=>lifetime_dynpro**: The control remains alive for the lifetime of the screen instance, that is, for as long as the screen remains in the stack. For further information, refer to Lifetime Management [Ext.].
parent **TYPE REF TO CL_GUI_CONTAINE R**		The container control [Ext.] in which you want the tree to appear
control **TYPE REF TO CL_GUI_CONTROL**		A reference to the tree control instance that the method creates

set_has_3d_frame

Use this method to specify whether the tree control should be displayed "flat" or with a 3-dimensional frame.

```
CALL METHOD model->set_has_3d_frame
          EXPORTING has_3d_frame = has_3d_frame.
```

Parameter and Type	Opt.	Description
has_3d_frame **TYPE AS4FLAG**		Flag indicating the frame type: • 'X': Tree is displayed with three-dimensional frame • ' ': Tree does not have a three-dimensional frame

update_view

This method synchronizes the tree model on the application server with its associated tree control instance at the frontend.

⚠

You should not have to use this method, since the tree model and tree display are synchronized automatically at the **end of each PBO event** and at the **end of each system event**.

```
CALL METHOD model->update_view.
```

add_key_stroke

Use this method to define keys that trigger an event. To react to the events, you must also register the corresponding event (**NODE_KEYPRESS** and/or **ITEM_KEYPRESS**).

```
CALL METHOD model->add_key_stroke
     EXPORTING  key = key.
```

Parameters	Description
key	Key that you want to trigger the event: **CL_TREE_MODEL=>KEY_F1**: Function key F1 **CL_TREE_MODEL=>KEY_F4**: Function key F4 **CL_TREE_MODEL=>KEY_INSERT**: Insert key **CL_TREE_MODEL=>KEY_DELETE**: Delete key

remove_all_key_strokes

Use this method to deregister all of the keystrokes that were registered to trigger the **`NODE_KEYPRESS`** and **`ITEM_KEYPRESS`** events.

```
CALL METHOD model->remove_all_key_strokes.
```

get_key_strokes

Use this method to find out which keys are registered to trigger the **KEYPRESS** event.

```
CALL METHOD model->get_key_strokes
          IMPORTING keystrokes = keystrokes.
```

Parameter and Type	Opt.	Description
keystrokes **TYPE TREEMINTEG**		A table containing the registered keys. For further information, refer to add_key_stroke [Page 262]. The internal table has the type **I**.

set_selected_node

Use this method to select a node in the tree model.

```
CALL METHOD model->set_selected_node
          EXPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node you want to select

select_nodes

Use this method to select a set of nodes in the tree model. You can only use it if the tree model instance supports multiple node selection.

```
CALL METHOD model->select_nodes
          EXPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
node_key_table TYPE **TREEMNOTAB**		An internal table, each line of which contains the key of a node you want to select. The table has the line type **TM_NODEKEY**.

unselect_all

Use this method to deselect all selected nodes.

```
CALL METHOD model->unselect all.
```

unselect_nodes

Use this method to deselect a set of nodes in the tree.

```
CALL METHOD model->unselect_nodes
          EXPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
node_key_table **TYPE TREEMNOTAB**		An internal table, each line of which contains the key of a node you want to deselect within the tree. The table has the line type **TM_NODEKEY**.

get_selected_node

Use this method to find out which node is currently selected.

Prerequisites

- You must have set up the tree model instance to support single node selection only
- You must have created a tree control in which the tree model can be displayed (create_tree_control [Page 259] method)

Syntax

```
CALL METHOD model->get_selected_node
          IMPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the selected node. If no node is selected, **node_key** has the value ' '.

get_selected_nodes

Use this method to find out which nodes are currently selected within the tree model.

Prerequisites

- You must have set up the tree model instance to support single node selection only
- You must have created a tree control in which the tree model can be displayed (create_tree_control [Page 259] method)

Syntax

```
CALL METHOD model->get_selected_nodes
          IMPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
node_key **TYPE TREEMNOTAB**		An internal table in which each line contains the key of the selected node. If no nodes are selected, the table is empty. The table has the line type **TM_NODEKEY**.

get_node_selection_mode

Use this method to find out whether single or multiple node selection has been set for the tree model instance.

```
CALL METHOD model->get_node_selection_mode
          IMPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_selection_mode **TYPE I**		The selection mode set up for the tree model instance. Possible values: • **cl_tree_model=>node_sel_mode_single**: Single node selection • **cl_tree_model=>node_sel_mode_multiple**: Multiple node selection

get_hide_selection

Use this method to find out the current setting of the **hide_selection** attribute for the tree model instance.

```
CALL METHOD model->get_hide_selection
          IMPORTING hide_selection = hide_selection.
```

Parameter and Type	Opt.	Description
hide_selection **TYPE AS4FLAG**		Current setting of the hide_selection attribute. Possible values: • **'X'**: Selection is hidden • **' '**: Selection is visible

node_keys_in_tree

Use this method to check whether the keys of a set of nodes are already used in the tree model instance. Duplicate keys are **not allowed**, and will lead to a **runtime error**. The SAP Tree Model, unlike the normal SAP Tree, provides you with this means of checking that you do not use duplicate keys.

```
CALL METHOD model->node_keys_in_tree
          EXPORTING node_key_table        = node_key_table
          IMPORTING node_keys_in_tree     = node_keys_in_tree
                    node_keys_not_in_tree = node_keys_not_in_tree.
```

Parameter and Type	Opt.	Description
node_key_table **TYPE TREEMNOTAB**		Internal table containing the node keys you want to check
node_keys_in_tree **TYPE TREEMNOTAB**		The nodes from **node_key_table** that are already contained in the tree model
node_keys_not_in_tree **TYPE TREEMNOTAB**		The ndoes from **node_key_table** that are not contained in the tree model

The internal table type **TREEMNOTAB** has the line type **TM_NODEKEY**.

node_key_in_tree

Use this method to find out whether a particular node key is already used in the tree model instance. Duplicate node keys are **not allowed** and lead to a **runtime error**.

```
CALL METHOD model->node_key_in_tree
          EXPORTING    node_key = node_key
          RETURNING key_in_tree = key_in_tree.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Node key that you want to check
key_in_tree **TYPE AS4FLAG**		Flag to indicate whether the key is already used in the tree model instance. Possible values: • **'X'**: Already used • **' '**: Not used

Since **key_in_tree** is the only returning parameter, you can evaluate it by writing the method call directly into an **IF** statement. For example:

```
IF model->node_key_in_tree( node_key = newnode ) = 'X'.
MESSAGE i999.
* Node already exists in tree
ELSE.
CALL METHOD model->add_node...
...
ENDIF.
```

expand_node

Use this method to expand a given node. The node must be a folder. You can also specify whether to expand its predecessor nodes (the nodes between it and the root node), and whether to expand its child nodes.

```
CALL METHOD model->expand_node
          EXPORTING node_key            = node_key
                    expand_predecessors = expand_predecessors
                    expand_subtree      = expand_subtree
                    level_count         = level_count.
```

Parameter and Type	Opt.	Description
node_key `TYPE TM_NODEKEY`		The key of the node you want to expand
expand_predecessors `TYPE AS4FLAG`	X	Flag indicating whether you want to expand the predecessor nodes.
expand_subtree `TYPE AS4FLAG`	X	Flag indicating whether you want to expand the child nodes. If it has the value 'X', the entire subtree below the node is expanded, regardless of the value you specify in `level_count`.
level_count `TYPE I`	X	Depth to which you want to expand the child nodes. Possible values: • `'0'`: Only the current node is expanded • `'1'`: The current node and its immediate successors are expanded • `'n'`: The current node and its successors down to the nth level are expanded. Note: If you specify `expand_subtree = 'X'`, the method ignores `level_count` and expands the entire subtree.

Expanding nodes can lead to large numbers of child nodes being transferred to the frontend control, which can lead to network timeouts. If you need to expand a lot of nodes, use the methods save_expand_all_nodes [Page 278] or save_expand_subree [Page 279].

expand_nodes

Use this method to expand a set of nodes in the tree. The node can only be expanded if they are folders containing child nodes.

```
CALL METHOD model->expand_nodes
          EXPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
node_key_table **TYPE TREEMNOTAB**		Internal table containing the keys of the nodes you want to expand. The table has the line type **TM_NODEKEY**.

➡

Expanding nodes can lead to large numbers of child nodes being transferred to the frontend control, which can lead to network timeouts. If you need to expand a lot of nodes, use the methods save_expand_all_nodes [Page 278] or save_expand_subree [Page 279].

expand_root_nodes

Use this method to expand all of the root nodes in the tree model. A root node has the attribute **RELATKEY = ' '**.

```
CALL METHOD model->expand_root_nodes
          EXPORTING expand_subtree = expand_subtree
                    level_count    = level_count.
```

Parameter and Type	Opt.	Description
expand_subtree **TYPE AS4FLAG**	X	If you set this parameter to **'X'**, the system expands all subtrees of the root nodes
level_count **TYPE I**	X	Specifies the depth to which the root nodes should be expanded. Possible values: • **0**: Only the root nodes themselves are expanded • **1**: The root nodes and their first level of child nodes are expanded • **n**: The root nodes are expanded down to their nth level of child nodes. Note: If you set the **expand_subtree** parameter to **'X'**, the value of **level_count** is ignored.

➡

Expanding nodes can lead to large numbers of child nodes being transferred to the frontend control, which can lead to network timeouts. If you need to expand a lot of nodes, use the methods save_expand_all_nodes [Page 278] or save_expand_subree [Page 279].

save_expand_all_nodes

Use this method to expand all of the nodes in the tree model instance. Since expanding all of the nodes may involve sending a very large number of nodes to the frontend control, the method does not necessarily attempt the expansion in a single step. Instead, it transfers as many nodes as it can without risking a network timeout. The tree is expanded by level, that is, first all root nodes, then all the child nodes of root nodes, and so on. The **all_nodes_expanded** parameter indicates whether all nodes could be transferred during the operation.

```
CALL METHOD model->save_expand_all_nodes
          IMPORTING all_nodes_expanded = all_nodes_expanded.
```

Parameter and Type	Opt.	Description
all_nodes_expanded **TYPE AS4FLAG**		Flag indicating whether all nodes could be expanded: • **'X'**: Yes • **' '**: No

save_expand_subree

Use this method to expand all of the nodes subordinate to a given node in the tree model instance. Since this may involve sending a very large number of nodes to the frontend control, the method does not necessarily attempt the expansion in a single step. Instead, it transfers as many nodes as it can without risking a network timeout. The tree is expanded by level, that is, first the node **node_key**, then all of its child nodes, and so on. The **all_nodes_expanded** parameter indicates whether all nodes could be transferred during the operation.

```
CALL METHOD model->save_expand_subtree
          EXPORTING node_key           = node_key
          IMPORTING all_nodes_expanded = all_nodes_expanded.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node you want to expand
all_nodes_expanded **TYPE AS4FLAG**		Flag indicating whether all of the nodes could be expanded: • **'X'**: Yes • **' '**: No

expand_node_predecessors

Use this method to expand all of the predecessor nodes of a given node, that is, all of the nodes between it and the root node.

```
CALL METHOD model->expand_node_predecessors
          EXPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		The key of the node whose predecessor nodes you want to expand

get_expanded_nodes

Use this method to find out the keys of all of the nodes in the tree model instance that are currently expanded.

```
CALL METHOD model->get_expanded_nodes
          EXPORTING no_hidden_nodes
          IMPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
no_hidden_nodes **TYPE AS4FLAG**	X	If you set this option, a node only counts as expanded if its predecessors are expanded.
node_key_table **TYPE TREEMNOTAB**		Internal table containing the keys of the expanded nodes. The table has the line type **TM_NODEKEY**.

collapse_all_nodes

Use this method to collapse the entire tree display.

```
CALL METHOD model->collapse_all_nodes.
```

collapse_node

Use this method to collapse a node in the tree. You can choose whether to collapse its subtree as well.

```
CALL METHOD model->collapse_node
          EXPORTING node_key         = node_key
                    collapse_subtree = collapse_subtree.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		The key of the node you want to collapse
collapse_subtree **TYPE AS4FLAG**	X	Indicates whether the child nodes of the specified node should also be collapsed: • **'X'**: Yes • **' '**: No (default)

get_first_root_node

Use this method to find out the key of the first root node in the tree model instance.

```
CALL METHOD model->get_first_root_node
          IMPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key `TYPE TM_NODEKEY`		The key of the first root node.

get_last_root_node

Use this method to find out the key of the last root node in the tree model instance.

```
CALL METHOD model->get_last_root_node
          IMPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the last root node.

get_nr_of_root_nodes

Use this method to find out how many root nodes there are in the tree model instance.

```
CALL METHOD model->get_nr_of_root_nodes
          RETURNING nr_of_root_nodes = nr_of_root_nodes.
```

Parameter and Type	Opt.	Description
nr_of_root_nodes **TYPE I**		Number of root nodes in the tree model instance

get_root_nodes

Use this method to get the keys of all of the root nodes in the tree model instance.

```
CALL METHOD model->get_root_nodes
          IMPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
node_key_table **TYPE TREEMNOTAB**		An internal table, each line of which contains the key of one root node. The table has the line type TM_NODEKEY.

delete_all_nodes

Use this method to delete all of the nodes from an instance of the tree model.

```
CALL METHOD model->delete_all_nodes.
```

delete_node

Use this method to delete a single node from the tree. It also deletes all of its child nodes.

```
CALL METHOD model->delete_node
          EXPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node you want to delete.

delete_nodes

Use this method to delete a set of nodes from the tree. All of their child nodes are also deleted.

```
CALL METHOD model->delete_nodes
          EXPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
node_key_table **TYPE TREEMNOTAB**		An internal table, each line of which contains the key of a node you want to delete.

node_set_disabled

Use this method to set the **disabled** attribute of a node. Disabled nodes cannot be selected.

```
CALL METHOD model->node_set_disabled
          EXPORTING node_key = node_key
                    disabled = disabled.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		The key of the node whose disabled attribute you want to change.
disabled **TYPE AS4FLAG**		The new value of the disabled parameter: • **'X'**: Node is disabled • **' '**: Node is not disabled

node_set_dragdropid

Use this method to set the drag and drop behavior of a node.

```
CALL METHOD model->node_set_dragdropid
          EXPORTING node_key    = node_key
                    dragdropid = dragdropid.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		The key of the relevant node
dragdropid **TYPE I**		The drag and drop [Page 106] behavior that you want to assign to the node.

node_set_expander

Use this method to set the **expander** attribute of a node. If you apply the method to a leaf, it is turned into a folder.

```
CALL METHOD model->node_set_expander
          EXPORTING node_key = node_key
                    expander = expander
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		The key of the node for which you want to change the attribute
expander **TYPE AS4FLAG**		The value of the **expander** attribute: • **'X'**: Sets the **expander** attribute • **' '**: Revokes the **expander** attribute.

node_set_expanded_image

Use this method to set the image that will appear to represent a folder when it is expanded.

```
CALL METHOD model->node_set_expanded_image
          EXPORTING node_key  = node_key
                    exp_image = exp_image.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node whose image you want to set
exp_image **TYPE TV_IMAGE**		The new image. Possible values: • `'  '`: No icon • `'@XY@'`: The SAP icon with code XY • `'BNONE'`: No icon. In a simple tree model, the node text then appears where the icon would normally appear. For ergonomic reasons, if you use this setting for a node, you should also use it for all nodes at the same level.

node_set_hidden

Use this method to set the hidden attribute of a node.

```
CALL METHOD model->node_set_hidden
          EXPORTING node_key = node_key
                    hidden   = hidden
```

Parameter and Type	Opt.	Description
node_key `TYPE TM_NODEKEY`		Key of the node whose attribute you want to change
hidden `TYPE AS4FLAG`		Sets the attribute: • `'X'`: Node is hidden • `'  '`: Node is visible

node_set_is_folder

Use this method to set the `is_folder` attribute for a node.

```
CALL METHOD model->node_set_is_folder
          EXPORTING node_key  = node_key
                    is_folder = is_folder.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node whose attribute you want to change
is_folder **TYPE AS4FLAG**		The new setting for the attribute: • `'X'`: Node is a folder • `' '`: Node is a leaf

➡

You can only convert a node to a leaf if it has no child nodes.

node_set_no_branch

Use this method to specify whether hierarchy lines should be drawn to a node.

```
CALL METHOD model->node_set_no_branch
          EXPORTING  node_key = node_key
                    no_branch = no_branch.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		The key of the node whose attribute you want to change
no_branch **TYPE AS4FLAG**		Indicates whether the hierarchy lines should be drawn to the node: • `'X'`: Yes • `' '`: No

node_set_image

Use this method to set the image that will appear to represent a leaf or closed folder.

```
CALL METHOD model->node_set_image
          EXPORTING node_key  = node_key
                    image     = image.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node whose image you want to set
image **TYPE TV_IMAGE**		The new image. Possible values: • `'  '`: No icon • `'@XY@'`: The SAP icon with code XY • `'BNONE'`: No icon. In a simple tree model, the node text then appears where the icon would normally appear. For ergonomic reasons, if you use this setting for a node, you should also use it for all nodes at the same level.

node_set_style

Use this method to set the style of a node.

```
CALL METHOD model->node_set_style
          EXPORTING node_key  = node_key
                    style     = style
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node for which you want to set the style
style **TYPE I**		The style of the item. Possible values: • `cl_tree_model=>style_inherited`: This has the same effect as `style_default` • `cl_tree_model=>style_default`: The item has the default text and background colors • `cl_tree_model=>style_intensified` • `cl_tree_model=>style_inactive` • `cl_tree_model=>style_intensified_critical` • `cl_tree_model=>style_emphasized_negative` • `cl_tree_model=>style_emphasized_positive` • `cl_tree_model=>style_emphasized`

node_set_user_object

Use this method to assign any object reference to the node. The reference can, for example, point to an object containing key application data relevant to the node.

```
CALL METHOD model->set_user_object
          EXPORTING node_key    = node_key
                    user_object = user_object.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node to which you want to assign the object
user_object **TYPE REF TO OBJECT**		Object reference

node_get_children

Use this method to get a list of the child nodes of a specified node.

```
CALL METHOD model->node_get_children
          EXPORTING node_key       = node_key
          IMPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		Key of the relevant node
node_key_table **TYPE** **TREEMNOTAB**		Internal table, each line of which contains the key of a child of the node specified in **node_key**. They are listed in the internal table in the order in which they occur in the tree.

node_get_first_child

Use this method to find out the first child node of a given node.

```
CALL METHOD model->node_get_first_child
          EXPORTING node_key       = node_key
          IMPORTING child_node_key = child_node_key.
```

Parameter and Type	Opt.	Description
node_key `TYPE TM_NODEKEY`		Key of the node whose child node you want to find
child_node_key `TYPE TM_NODEKEY`		The node key of the first child node.

node_get_last_child

Use this method to find out the last child node of a given node.

```
CALL METHOD model->node_get_last_child
          EXPORTING node_key       = node_key
          IMPORTING child_node_key = child_node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node whose child node you want to find
child_node_key **TYPE TM_NODEKEY**		The node key of the last child node.

node_get_next_sibling

Use this method to find out the key of the next node at the same level as a given node.

```
CALL METHOD model->node_get_next_sibling
          EXPORTING node_key         = node_key
          IMPORTING sibling_node_key = sibling_node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node to which the item belongs
sibling_node_key **TYPE TM_NODEKEY**		The node key of the next same-level node

node_get_nr_of_children

Use this method to find out how many child nodes a given node has.

```
CALL METHOD model->node_get_nr_of_children
          EXPORTING node_key        = node_key
          IMPORTING nr_of_children = nr_of_children.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node
nr_of_children **TYPE I**		The number of child nodes that the node has

node_get_parent

Use this method to find out the key of the parent node of a given node.

```
CALL METHOD model->node_get_parent
          EXPORTING node_key        = node_key
          IMPORTING parent_node_key = parent_node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node
parent_node_key **TYPE TM_NODEKEY**		The node key of the parent node

node_get_prev_sibling

Use this method to find out the key of the previous node at the same level as a given node.

```
CALL METHOD model->node_get_next_sibling
          EXPORTING node_key         = node_key
          IMPORTING sibling_node_key = sibling_node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node to which the item belongs
sibling_node_key **TYPE TM_NODEKEY**		The node key of the previous same-level node

node_get_user_object

Use this method to retrieve the user object of a given node.

```
CALL METHOD model->node_get_user_object
          EXPORTING node_key    = node_key
          IMPORTING user_object = user_object.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node
user_object **TYPE REF TO OBJECT**		The user object belonging to the node

print_tree

Use this method to print the contents of the tree.

```
CALL METHOD model->print_tree
          EXPORTING all_nodes = all_nodes
                    title     = title
                    preview   = preview.
```

Parameter and Type	Opt.	Description
all_nodes **TYPE AS4FLAG**		Indicates how much of the tree should be printed: • 'X': Print the whole tree • ' ': Only print expanded nodes
title **TYPE STRING**	X	Title to be printed with the tree
preview **TYPE AS4FLAG**		Flag indicating whether a print preview should be displayed before the tree is printed: • 'X': Display preview • ' ': No preview

get_nr_of_nodes

Use this method to find out the number of nodes in the tree model.

```
CALL METHOD model->get_nr_of_nodes
          IMPORTING nr_of_nodes = nr_of_nodes.
```

Parameter and Type	Opt.	Description
nr_of_nodes **TYPE I**		The number of nodes in the tree

ensure_visible

Use this method to ensure that a specified node appears within the tree display at the frontend.

```
CALL METHOD model->ensure_visible
          EXPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key TYPE TM_NODEKEY		Key of the node that should be visible

move_node

Use this method to move a node within the tree model.

```
CALL METHOD model->move_node
          EXPORTING node_key          = node_key
                    relative_node_key = relative_node_key
                    relationship      = relationship.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		Key of the node you want to move
relative_node_ke y **TYPE** **TM_NODEKEY**		Key of the node to which the node will be related in its new position
relationship **TYPE I**		The relationship between the new node and the node specified in **relative_node_key**. Possible values are: • **CL_TREE_MODEL=>RELAT_FIRST_CHILD** Inserts the new node as the first child node of the node specified in **relative_node_key**. This must be a folder. • **CL_TREE_MODEL=>RELAT_LAST_CHILD** Inserts the new node as the last child node of the node specified in **relative_node_key**. This must be a folder. • **CL_TREE_MODEL=>RELAT_PREV_SIBLING** Inserts the new node directly before the related node at the same level. • **CL_TREE_MODEL=>RELAT_NEXT_SIBLING** Inserts the new node directly after the related node at the same level. • **CL_TREE_MODEL=>RELAT_FIRST_SIBLING** Inserts the new node as the first node at the same level as the related node. • **CL_TREE_MODEL=>RELAT_LAST_SIBLING** Inserts the new node as the last node at the same level as the related node. Note: If **relative_node_key** is empty, the new node is inserted as a root node. Where the above values contain the word **FIRST** or **PREV**, it is inserted as the first root node. Where they contain **LAST** or **NEXT**, it is inserted as the last.

scroll

Use this method to scroll the tree control display.

```
CALL METHOD model->scroll
          EXPORTING scroll_command = scroll_command.
```

Parameter and Type	Opt.	Description
scroll_command **TYPE I**		Specification of the scroll operation. Possible values: • **cl_tree_model=>scroll_up_line**: Scrolls up one line • **cl_tree_model=>scroll_down_line**: Scrolls down one line • **cl_tree_model=>scroll_up_page**: Scrolls up one page • **cl_tree_model=>scroll_down_page**: Scrolls down one page • **cl_tree_model=>scroll_home**: Scrolls to the top of the tree • **cl_tree_model=>scroll_end**: Scrolls to the bottom of the tree

set_ctx_menu_select_event_appl

Use this method to specify whether the **SELECT** event of a context menu should be an application event (triggers PAI) or a system event (does not trigger PAI). You must call this event before **set_registered_events**.

```
CALL METHOD model->set_ctx_menu_select_event_appl
          EXPORTING appl_event = appl_event.
```

Parameter and Type	Opt.	Description
appl_event **TYPE AS4FLAG**		Specifies whether the event should be an application event: • 'X': Application event • ' ': System event

get_ctx_menu_select_event_appl

Use this method to find out whether the SELECT event for context menus is registered as an application event or a system event.

```
CALL METHOD model->get_ctx_menu_select_event_appl
          IMPORTING appl_event = appl_event.
```

Parameter and Type	Opt.	Description
appl_event TYPE AS4FLAG		Specifies whether the event is an application event: • 'X': Application event • ' ': System event

set_default_drop

Use this method to set the default drop behavior for drag and drop operations in which the object is dropped onto the control area.

```
CALL METHOD model->set_default_drop
          EXPORTING drag_drop = drag_drop.
```

Parameter and Type	Opt.	Description
drag_drop **TYPE REF TO CL_DRAGDROP**		The default description

get_default_drop

Use this method to find out the default drop behavior for drag and drop operations in which the object is dropped onto the control area.

```
CALL METHOD model->get_default_drop
          IMPORTING drag_drop = drag_drop.
```

Parameter and Type	Opt.	Description
drag_drop **TYPE REF TO CL_DRAGDROP**		The default description

set_folder_show_exp_image

Use this method to select which icon will be displayed when a folder is expanded.

```
CALL METHOD model->set_folder_show_exp_image
          EXPORTING folder_show_exp_image = folder_show_exp_image.
```

Parameter and Type	Opt.	Description
folder_show_exp_image **TYPE AS4FLAG**		Flag specifying which icon will be displayed: • **'X'**: The icon specified in the **EXPANDED_IMAGE** attribute is always displayed when the folder is expanded • **' '**: The icon specified in the **EXPANDED_IMAGE** attribute is only displayed for the last folder to be expanded. All other open folders display the icon specified in the **IMAGE** attribute.

For further information, refer to node_set_image [Page 298] and node_set_expanded_image [Page 294].

get_folder_show_exp_image

Use this method to find out which icon is displayed when a folder is expanded.

```
CALL METHOD model->get_folder_show_exp_image
          IMPORTING folder_show_exp_image = folder_show_exp_image.
```

Parameter and Type	Opt.	Description
folder_show_exp_image **TYPE AS4FLAG**		Flag specifying which icon will be displayed: • **'X'**: The icon specified in the **EXPANDED_IMAGE** attribute is always displayed when the folder is expanded • **' '**: The icon specified in the **EXPANDED_IMAGE** attribute is only displayed for the last folder to be expanded. All other open folders display the icon specified in the **IMAGE** attribute.

set_top_node

Use this method to set the topmost node in the tree display.

```
CALL METHOD model->set_top_node
          EXPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key TYPE TM_NODEKEY		The key of the node you want to appear at the top of the display

get_top_node

Use this method to find the topmost node in the tree display.

```
CALL METHOD model->set_top_node
          IMPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node at the top of the display

get_first_root_node

Use this method to find out the first root node in the tree.

```
CALL METHOD model->get_first_root_node
          IMPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key TYPE TM_NODEKEY		The key of the first root node in the tree

get_last_root_node

Use this method to get the key of the last root node in the tree.

```
CALL METHOD model->get_last_root_node
          IMPORTING node_key = node_key.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the last root node in the tree

Methods of Class CL_SIMPLE_TREE_MODEL

constructor

The constructor method is called automatically when you instantiate the class **cl_simple_tree_model**. To do this, you must declare a reference variable as follows:

```
DATA simple_model TYPE REF TO cl_simple_tree_model.
```

You can then create an instance using the **CREATE OBJECT** statement.

```
CREATE OBJECT simple_model
      EXPORTING node_selection_mode = node_selection_mode
                hide_selection      = hide_selection.
```

Parameter	Opt.	Description
node_selection_mode		Specifies whether or not multiple nodes can be selected simultaneously. Possible values are • **cl_simple_tree_model=>node_sel_mode_single** Only one node at a time may be selected • **cl_simple_tree_model=>node_sel_mode_multiple** Multiple nodes may be selected
hide_selection	X	Specifies whether the selection should be hidden. Possible values are • **'X'** - hide selection • **' '** - Show selection

add_node

Use this method to add a single node to the simple tree model. The node is inserted in the tree structure within the model, and transported to the visible tree at the frontend at the end of the next PBO event.

```
CALL METHOD simple_model->add_node
      EXPORTING node_key          = node_key
                relative_node_key = relative_node_key
                relationship      = relationship
                isfolder          = isfolder
                text              = text
                hidden            = hidden
                disabled          = disabled
                style             = style
                no_branch         = no_branch
                expander          = expander
                image             = image
                expanded_image    = expanded_image
                drag_drop_object  = drag_drop_object
                user_object       = user_object.
```

Parameter and Type	Opt.	Description
node_key `TYPE STRING`		The key by which the node is identified in the tree. **This must be unique throughout the tree**. You should only use letters, digits, and the underscore character in node keys.
relative_node_key `TYPE STRING`	X	The key of a node to which the new node is related in position. If the new node is the first or last root node, this parameter must have the value `' '`.

relationship **I**	X	The relationship between the new node and the node specified in **relative_node_key**. Possible values are: • **CL_TREE_MODEL=>RELAT_FIRST_CHILD** Inserts the new node as the first child node of the node specified in **relative_node_key**. This must be a folder. • **CL_TREE_MODEL=>RELAT_LAST_CHILD** Inserts the new node as the last child node of the node specified in **relative_node_key**. This must be a folder. • **CL_TREE_MODEL=>RELAT_PREV_SIBLING** Inserts the new node directly before the related node at the same level. • **CL_TREE_MODEL=>RELAT_NEXT_SIBLING** Inserts the new node directly after the related node at the same level. • **CL_TREE_MODEL=>RELAT_FIRST_SIBLING** Inserts the new node as the first node at the same level as the related node. • **CL_TREE_MODEL=>RELAT_LAST_SIBLING** Inserts the new node as the last node at the same level as the related node. Note: If **relative_node_key** is empty, the new node is inserted as a root node. Where the above values contain the word **FIRST** or **PREV**, it is inserted as the first root node. Where they contain **LAST** or **NEXT**, it is inserted as the last.
isfolder **TYPE AS4FLAG**		Specifies whether the new node should be a folder or a leaf. Possible values: • **'X'**: Node is a folder • **' '**: Node is a leaf
text **TYPE STRING**		Text for the node
hidden **TYPE AS4FLAG**	X	Specifies whether the node should be hidden (**'X'**) or visible (**' '**). Default is visible.
disabled **TYPE AS4FLAG**	X	Specifies whether the node can be selected (**' '**) or not (**'X'**). The default is **not** disabled. **Note**: If a node is disabled, actions such as double-clicking it have no effect.
style **TYPE AS4FLAG**	X	Sets the colors of the text and the background for the node. The possible values for this field are any static constant **CL_TREE_MODEL=>STYLE_***. For further details, refer to the definition of **CL_TREE_MODEL** in the Class Builder.

no_branch **TYPE AS4FLAG**	X	Specifies whether connecting lines should be drawn between the nodes (`' '`) or not (**`'X'`**). The default is for the lines to be drawn.
expander **TYPE AS4FLAG**	X	May only be set for a folder. If you set this attribute, the closed folder always displays a '+' symbol, even if it is empty. When the user clicks on the folder, the event **EXPAND_NO_CHILDREN** is triggered.
image **C(6)**	X	Specifies the image used for the node. Possible values: • initial: The system uses the default values (leaf symbol for a leaf, closed folder symbol for a folder) • **'@XY@'**: An SAP icon with the code **XY**. • **'BNONE'**: No image is displayed. The node text begins at the position in which the image would normally be displayed. If you use this value for a node, you should also use it for all of its other same-level nodes.
expanded_image **C(6)**	X	Specifies the image used for an open folder. The possible values are the same as those listed above for the *image* parameter.
drag_drop_object **I**	X	Only relevant if you want the node to be drag and drop-enabled. It contains the handle for a drag and drop object.
user_object **TYPE REF TO OBJECT**	X	Can be assigned any reference to an application object

add_nodes

Use this method to add a set of nodes to the tree model. The nodes are inserted into the tree structure within the model, and transported to the visible tree at the frontend at the end of the next PBO event.

```
CALL METHOD simple_model->add_nodes
      EXPORTING node_table = node_table.
```

Parameter and Type	Opt.	Description
node_table **TYPE TREEMSNOTA**		Internal table containing the nodes you want to add to the tree model. Each row of the table represents a node. The data type **TREEMSNOTA** has the line type TREEMSNODT [Page 451].

update_nodes

Use this method to change the attributes of nodes in the tree model. You cannot use it to change the **RELATKEY** or **RELATSHIP** attributes. If you want to move a node, use the **MOVE_NODE** method.

```
CALL METHOD simple_model->update_nodes
      EXPORTING node_table = node_table.
```

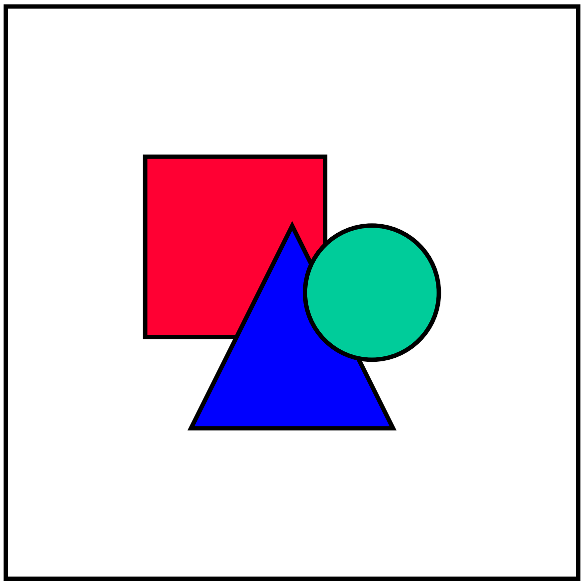
Parameter and Type	Opt.	Description
node_table **TYPE TREEMSUNOT**		An internal table in which each line represents one node whose attributes you want to change. You specify the key of the node, and enter a new value for each attribute that you want to change. Each changeable attribute also has a corresponding flag with the name **U_<attribute>**. You must check this flag for each attribute that you change. For example, if you want to change the hidden attribute for a node from `' '` (not hidden) to **'X'** (hidden), you would enter **'X'** in the **HIDDEN** field and **'X'** in the field **U_HIDDEN** (to indicate that the field must be updated). If you want to change all of the changeable attributes for a given node, you should check the **U_ALL** field instead of all of the individual **U_<attribute>** flag fields. The data type **TREEMSUNOT** has the line type TREEMSUNO [Page 454].

set_registered_events

Use this method to register a set of events with the Control Framework. For further information, refer to Processing Events in the Tree Model [Page 257].

```
CALL METHOD tree_model->set_registered_events
             EXPORTING events = events.
```

Parameter and Type	Opt.	Description
events **TYPE CNTL_SIMPLE_EVENTS**		Internal table in which each row represents an event that you want to register.



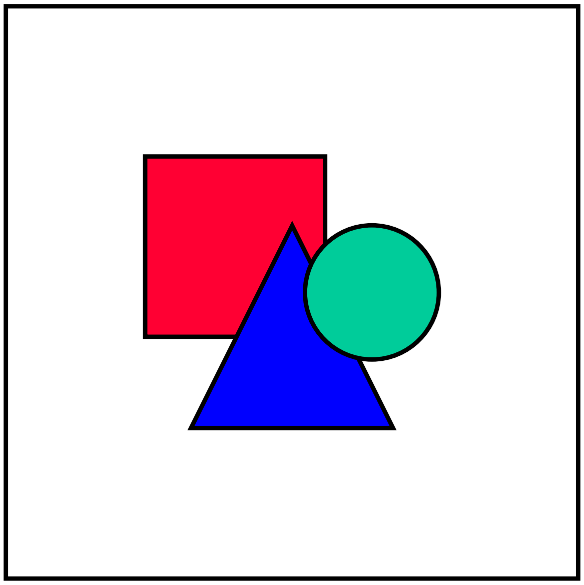
When you fill the internal table for the events parameter, you can use the data type `CNTL_SIMPLE_EVENT` to define a work area.

Structure of CNTL_SIMPLE_EVENT

Component	Type	Opt.	Description
eventid	`I`		The ID of the event you want to register. These are all defined as static constants in the relevant wrapper class. For further information, refer to Tree Model Events [Ext.].
appl_event			Specifies the type of the event: • `' '`: System event • `'X'`: Application event For further information, refer to Processing Events in the Tree Model [Page 257].

get_registered_events

Use this method to return a list of the events that are registered at the Control Framework for the control instance.



Events in this list are registered at the Control Framework. However, in order for the event to be handled, you must also have registered its handler method using the **SET HANDLER** statement.

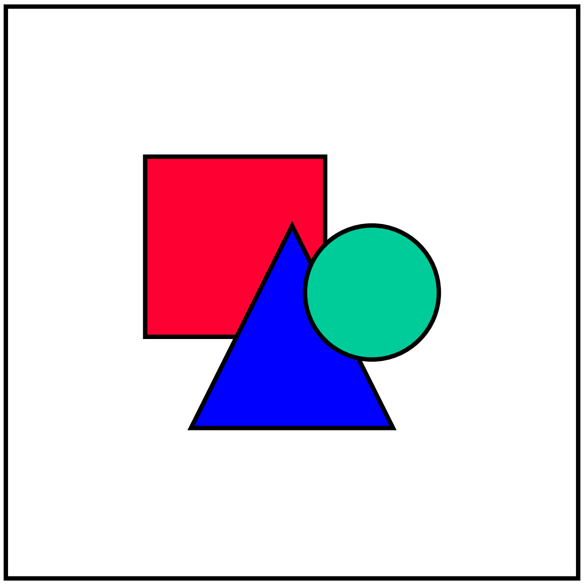
```
CALL METHOD tree_model->get_registered_events
             IMPORTING events = events.
```

Parameter and Type	Opt.	Description
events **TYPE CNTL_SIMPLE_EVENTS**		Internal table in which each row represents an event that you want to register.

node_set_text

Use this method to set the text for a node in the simple tree model.

```
CALL METHOD simple_tree->node_set_text
             EXPORTING node_key = node_key
                       text     =.text.
```



Although the **text** attribute is implemented with type **STRING**, and therefore has no defined maximum length, only the first 100 characters of the text can be displayed in the frontend control.

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		Key of the node for which you want to set the text
text **TYPE STRING**		New node text

node_get_text

Use this method to return the text of a node in the simple tree model.

```
CALL METHOD simple_tree->node_get_text
             EXPORTING node_key = node_key
             IMPORTING text     =.text.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		Key of the node for which you want to retrieve the text
text **TYPE STRING**		Node text

node_get_properties

Use this method to find out the properties of a given node.

```
CALL METHOD simple_tree->node_get_properties
             EXPORTING node_key    = node_key
             IMPORTING properties  = properties.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		Key of the node for which you want to set the text
properties **TYPE TREEMSNODT**		Properties of the node. For a full description of the structure of this parameter, refer to Structure TREEMSNODT [Page 451].

get_tree

Use this method to return the contents of the tree in an internal table.

```
CALL METHOD simple_model->get_tree
      IMPORTING node_table = node_table.
```

Parameter and Type	Opt.	Description
node_table **TYPE TREEMSNOTA**		Internal table containing the nodes in the tree model. Each row of the table represents a node. The data type **TREEMSNOTA** has the line type TREEMSNODT [Page 451].

find

Use this method to allow the user to search for a string within the tree. It displays a dialog box in which the user can enter the search string and specify whether to search specifically or using the ABAP operator CP.

```
CALL METHOD simple_model->find_first
         IMPORTING  result_type          = result_type
                    result_node_key      = result_node_key.
```

Parameter and Type	Opt.	Description
result_type **TYPE I**		This parameter contains the reason why the search stopped. It can have the following values: • **cl_simple_tree_model=>find_match** The search string was found. • **cl_simple_tree_model=>find_no_match** The search string was not found • **cl_simple_tree_model=>find_expander_node_hit** The search has encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. **Note**: This only applies if you set the **stop_at_expander_node** parameter.
result_node_key **TYPE STRING**		The key of the node at which the search stopped

find_first

Use this method to find the first occurrance of a string in the tree. The system searches the tree from top to bottom, starting from a node that you specify.

```
CALL METHOD simple_model->find_first
      EXPORTING search_string         = search_string
                pattern_search        = pattern_search
                start_node            = start_node
                stop_at_expander_node = stop_at_expander_node
      IMPORTING result_type           = result_type
                result_node_key       = result_node_key.
```

Parameter and Type	Opt.	Description
search_string **`TYPE TM_NODEKEY`**		The string for which you want to search
pattern_search **`TYPE AS4FLAG`**	X	Flag indicating whether you want to search specifically or generically. If you select this option, the search uses the ABAP operator CP (contains pattern)
start_node **`TYPE TM_NODEKEY`**	X	The starting node for the search. If you do not specify a starting node, the system searches from the root node of the tree
stop_at_expander_node **`TYPE AS4FLAG`**	X	If you set this option, the search stops if it encounters a node that has the attribute **`EXPANDER = 'X'`** but no child nodes. The **`result_type`** and **`result_node_key`** attributes are set accordingly. You can then load the child nodes into the tree model before resuming the search using the FIND_NEXT [Page 341] method.
result_type **`TYPE I`**		This parameter contains the reason why the search stopped. It can have the following values: • **`cl_simple_tree_model=>find_match`** `The search string was found.` • **`cl_simple_tree_model=>find_no_match`** The search string was not found • **`cl_simple_tree_model=>find_expander_node_hit`** The search has encountered a node with the attribute **`EXPANDER = 'X'`** and no child nodes. **Note**: This only applies if you set the **`stop_at_expander_node`** parameter.
result_node_key **`TYPE TM_NODEKEY`**		The key of the node at which the search stopped. **Caution**: If the search string was not found, this parameter is not filled. It may therefore be empty. However, it may also contain a value from a previous search.

find_next

You use this method n the following circumstances to resume a search that you started with the **find_first** method:

- To find the next occurrence of the search string
- To resume a search which terminated because it encountered a node with the attribute **EXPANDER = 'X'** and no child nodes.

It uses the same search criteria as you specified in **find_first**.

```
CALL METHOD simple_model->find_first
      IMPORTING result_type          = result_type
                result_node_key      = result_node_key.
```

Parameter and Type	Opt.	Description
result_type **TYPE I**		This parameter contains the reason why the search stopped. It can have the following values: • **cl_simple_tree_model=>find_match** The search string was found. • **cl_simple_tree_model=>find_no_match** The search string was not found • **cl_simple_tree_model=>find_expander_node_hit** The search has encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. **Note**: This only applies if you set the **stop_at_expander_node** parameter.
result_node_key **TYPE STRING**		The key of the node at which the search stopped. **Note**: If the search string was not found, this parameter is empty.

find_all

Use this method to search for all occurrences of a search string within the tree model.

```
CALL METHOD simple_model->find_all
      EXPORTING search_string            = search_string
                pattern_search           = pattern_search
                start_node               = start_node
                stop_at_expander_node    = stop_at_expander_node
                result_type              = result_type
                result_expander_node_key = result_expander_node_key
                result_node_key_table    = result_node_key_table
```

Parameter and Type	Opt.	Description
search_string **TYPE STRING**		The string for which you want to search
pattern_search **TYPE AS4FLAG**		Flag indicating whether you want to search specifically or generically. If you select this option, the search uses the ABAP operator CP (contains pattern)
start_node **TYPE STRING**		The starting node for the search. If you do not specify a starting node, the system searches from the root node of the tree
stop_at_expander_node **TYPE AS4FLAG**		If you set this option, the search stops if it encounters a node that has the attribute **EXPANDER = 'X'** but no child nodes. The **result_type** and **result_node_key** attributes are set accordingly. You can then load the child nodes into the tree model before resuming the search using the FIND_ALL_CONTINUE [Page 344] method.
result_type **TYPE I**		This parameter contains the reason why the search stopped. It can have the following values: • **cl_simple_tree_model=>find_match** The search string was found. • **cl_simple_tree_model=>find_no_match** The search string was not found • **cl_simple_tree_model=>find_expander_node_hit** The search has encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. **Note**: This only applies if you set the **stop_at_expander_node** parameter.
result_expander_node_key **TYPE STRING**		The key of the node at which the search stopped. **Note**: If the search string was not found, this parameter is empty.

result_node_key_table **TYPE TREEMNOTAB**		An internal table containing the node keys of the nodes in which the search string was found. The data type **TREEMNOTAB** is an internal table whose line type is a single **STRING** field.

find_all_continue

Use this method to resume a search that you started using the FIND_ALL [Page 342] method and which was interrupted because the search encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. The method uses the same search criteria as you used in the FIND_ALL method.

```
CALL METHOD simple_model->find_all_continue
      IMPORTING result_type              = result_type
                result_expander_node_key = result_expander_node_key
                result_node_key_table    = result_node_key_table.
```

Parameter and Type	Opt.	Description
result_type **TYPE I**		This parameter contains the reason why the search stopped. It can have the following values: • **cl_simple_tree_model=>find_match** The search string was found. • **cl_simple_tree_model=>find_no_match** The search string was not found • **cl_simple_tree_model=>find_expander_node_hit** The search has encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. **Note**: This only applies if you set the **stop_at_expander_node** parameter.
result_expander_node_key **TYPE STRING**		The key of the node at which the search stopped. **Note**: If the search string was not found, this parameter is empty.
result_node_key_table **TYPE TREEMNOTAB**		An internal table containing the node keys of the nodes in which the search string was found. The data type **TREEMNOTAB** is an internal table whose line type is a single **STRING** field.

Methods of Class CL_ITEM_TREE_MODEL

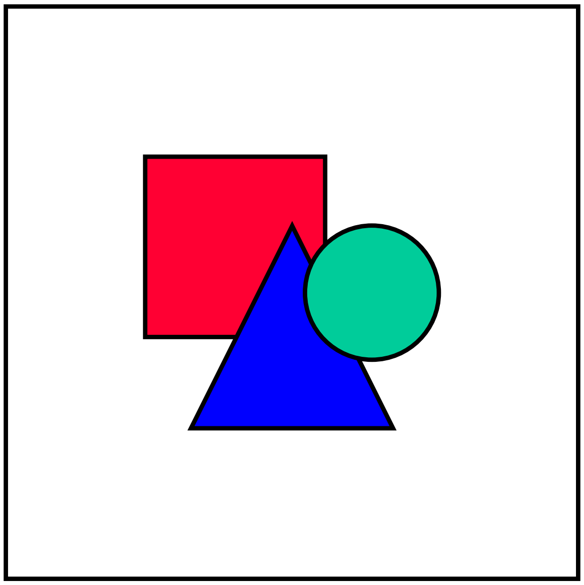
The class **CL_ITEM_TREE_MODEL** contains methods and attributes that are shared by the Column Tree Model and the List Tree Model. You can address its components as though they belonged to either **CL_COLUMN_TREE_MODEL** or **CL_LIST_TREE_MODEL**.

set_registered_events

Use this method to register a set of events with the Control Framework. For further information, refer to Processing Events in the Tree Model [Page 257].

```
CALL METHOD tree_model->set_registered_events
             EXPORTING events = events.
```

Parameter and Type	Opt.	Description
events **TYPE CNTL_SIMPLE_EVENTS**		Internal table in which each row represents an event that you want to register.



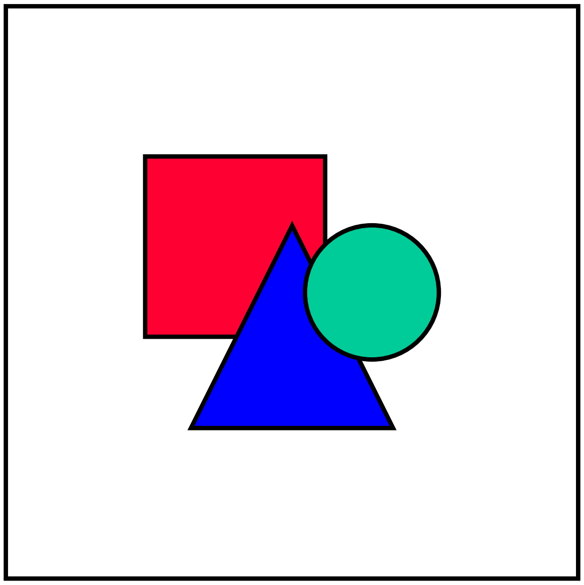
When you fill the internal table for the events parameter, you can use the data type **CNTL_SIMPLE_EVENT** to define a work area.

Structure of CNTL_SIMPLE_EVENT

Component	Type	Opt.	Description
eventid	**I**		The ID of the event you want to register. These are all defined as static constants in the relevant wrapper class. For further information, refer to Tree Model Events [Ext.].
appl_event			Specifies the type of the event: • `' '`: System event • `'X'`: Application event For further information, refer to Processing Events in the Tree Model [Page 257].

get_registered_events

Use this method to return a list of the events that are registered at the Control Framework for the control instance.



Events in this list are registered at the Control Framework. However, in order for the event to be handled, you must also have registered its handler method using the **SET HANDLER** statement.

```
CALL METHOD tree_model->get_registered_events
             IMPORTING events = events.
```

Parameter and Type	Opt.	Description
events **TYPE CNTL_SIMPLE_EVENTS**		Internal table in which each row represents an event that you want to register.

find

Use this method to allow the user to search for a string within the tree. It displays a dialog box in which the user can enter the search string a specify whether to search specifically or using the ABAP operator CP.

```
CALL METHOD item_model->find
          IMPORTING result_type           = result_type
                    result_item_key_table = result_item_key_table.
```

Parameter and Type	Opt.	Description
result_type **TYPE I**		The reason why the search stopped. It can have the following values: • **cl_item_tree_model=>find_match**: The search string was found • **cl_item_tree_model=>find_no_match**: The search string was not found
result_item_key_table **TYPE TREEMIKS**		An internal table, each line of which represents a point at which the search string was found. The line type has two fields: • **node_key** (type **TM_NODEKEY**) • **item_name** (type **TV_ITMNAME**)

find_first

Use this method to find the first occurrence of a string in the tree. The system searches the tree from top to bottom, starting at a node that you specify. You can also restrict the search to certain items within the tree structure.

```
CALL METHOD item_model->find_first
         EXPORTING search_string              = search_string
                   item_name_table            = item_name_table
                   pattern_search             = pattern_search
                   start_node                 = start_node
                   stop_at_expander_node      = stop_at_expander_node
         IMPORTING result_type                = result_type
                   result_item_key_table      = result_item_key_table
                   result_expander_node_key = result_expander_node_key.
```

Parameter and Type	Opt.	Description
search_string **TYPE TM_NODEKEY**		The string for which you want to search
item_name_table **TYPE TREEMINAMT**	X	You can use this internal table to specify the names of items. If you do, the system only searches for the search string in items whose names are contained in the table.
pattern_search **TYPE AS4FLAG**	X	Flag indicating whether you want to search specifically or generically. If you select this option, the search uses the ABAP operator CP (contains pattern)
start_node **TYPE TM_NODEKEY**	X	The starting node for the search. If you do not specify a starting node, the system searches from the root node of the tree
stop_at_expander_nod e **TYPE AS4FLAG**	X	If you set this option, the search stops if it encounters a node that has the attribute **EXPANDER = 'X'** but no child nodes. The **result_type** and **result_node_key** attributes are set accordingly. You can then load the child nodes into the tree model before resuming the search using the FIND_NEXT [Page 351] method.
result_type **TYPE I**		This parameter contains the reason why the search stopped. It can have the following values: • **cl_item_tree_model=>find_match** The search string was found. • **cl_item_tree_model=>find_no_match** The search string was not found • **cl_item_tree_model=>find_expander_node_hit** The search has encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. **Note**: This only applies if you set the **stop_at_expander_node** parameter.

result_item_key_table **TYPE TREEMIKS**		An internal table, each line of which represents a point at which the search string was found. The line type has two fields: • **node_key** (type **TM_NODEKEY**) • **item_name** (type **TV_ITMNAME**)
result_node_key **TYPE TM_NODEKEY**		The key of the node at which the search stopped. **Caution**: If the search string was not found, this parameter is not filled. It may therefore be empty. However, it may also contain a value from a previous search.

find_next

Use this method to resume a search that you started using the find_first [Page 349] method (for example, after the initial search stopped with result type **find_expander_node_hit**). The new search inherits the same search criteria as the initial search.

```
CALL METHOD item_model->find_next
          IMPORTING result_type             = result_type
                    result_item_key_table    = result_item_key_table
                    result_expander_node_key = result_expander_node_key.
```

Parameter and Type	Opt.	Description
result_type **TYPE I**		This parameter contains the reason why the search stopped. It can have the following values: • **cl_item_tree_model=>find_match** The search string was found. • **cl_item_tree_model=>find_no_match** The search string was not found • **cl_item_tree_model=>find_expander_node_hit** The search has encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. **Note**: This only applies if you set the **stop_at_expander_node** parameter.
result_item_key_table **TYPE TREEMIKS**		An internal table, each line of which represents a point at which the search string was found. The line type has two fields: • **node_key** (type **TM_NODEKEY**) • **item_name** (type **TV_ITMNAME**)
result_node_key **TYPE TM_NODEKEY**		The key of the node at which the search stopped. **Caution**: If the search string was not found, this parameter is not filled. It may therefore be empty. However, it may also contain a value from a previous search.

find_all

Use this method to find all occurrences of a search string within an item tree model instance.

```
CALL METHOD item_model->find_all
         EXPORTING search_string             = search_string
                   item_name_table           = item_name_table
                   pattern_search            = pattern_search
                   start_node                = start_node
                   stop_at_expander_node     = stop_at_expander_node
         IMPORTING result_type               = result_type
                   result_item_key_table     = result_item_key_table
                   result_expander_node_key = result_expander_node_key.
```

Parameter and Type	Opt.	Description
search_string **TYPE TM_NODEKEY**		The string for which you want to search
item_name_table **TYPE TREEMINAMT**	X	You can use this internal table to specify the names of items. If you do, the system only searches for the search string in items whose names are contained in the table.
pattern_search **TYPE AS4FLAG**	X	Flag indicating whether you want to search specifically or generically. If you select this option, the search uses the ABAP operator CP (contains pattern)
start_node **TYPE TM_NODEKEY**	X	The starting node for the search. If you do not specify a starting node, the system searches from the root node of the tree
stop_at_expander_nod e **TYPE AS4FLAG**	X	If you set this option, the search stops if it encounters a node that has the attribute **EXPANDER = 'X'** but no child nodes. The **result_type** and **result_node_key** attributes are set accordingly. You can then load the child nodes into the tree model before resuming the search using the FIND_ALL_CONTINUE [Page 354] method.
result_type **TYPE I**		This parameter contains the reason why the search stopped. It can have the following values: • **cl_item_tree_model=>find_match** The search string was found. • **cl_item_tree_model=>find_no_match** The search string was not found • **cl_item_tree_model=>find_expander_node_hit** The search has encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. **Note**: This only applies if you set the **stop_at_expander_node** parameter.

result_item_key_table **TYPE TREEMIKS**		An internal table, each line of which represents a point at which the search string was found. The line type has two fields: • **node_key** (type **TM_NODEKEY**) • **item_name** (type **TV_ITMNAME**)
result_node_key **TYPE TM_NODEKEY**		The key of the node at which the search stopped. **Caution**: If the search string was not found, this parameter is not filled. It may therefore be empty. However, it may also contain a value from a previous search.

find_all_continue

Use this method to resume a search that you started using the FIND_ALL [Page 352] method and which was interrupted because the search encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. The method uses the same search criteria as you used in the FIND_ALL method.

```
CALL METHOD item_model->find_all_continue
          IMPORTING result_type              = result_type
                   result_item_key_table    = result_item_key_table
                   result_expander_node_key = result_expander_node_key.
```

Parameter and Type	Opt.	Description
result_type **TYPE I**		This parameter contains the reason why the search stopped. It can have the following values: • **cl_item_tree_model=>find_match** The search string was found. • **cl_item_tree_model=>find_no_match** The search string was not found • **cl_item_tree_model=>find_expander_node_hit** The search has encountered a node with the attribute **EXPANDER = 'X'** and no child nodes. **Note**: This only applies if you set the **stop_at_expander_node** parameter.
result_item_key_table **TYPE TREEMIKS**		An internal table, each line of which represents a point at which the search string was found. The line type has two fields: • **node_key** (type **TM_NODEKEY**) • **item_name** (type **TV_ITMNAME**)
result_node_key **TYPE TM_NODEKEY**		The key of the node at which the search stopped. **Caution**: If the search string was not found, this parameter is not filled. It may therefore be empty. However, it may also contain a value from a previous search.

select_item

Use this method to select a specific item in an item tree model. Selecting an item using this method cancels any earlier selection the user may have made.

```
CALL METHOD item_model->select_item
          EXPORTING node_key  = node_key
                    item_name = item_name.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Node containing the item you want to select
item_name **TYPE TV_ITMNAME**		Item that you want to select

This method is subject to the usual rules concerning item selection (for example, an item with the attribute **DISABLED = 'X'** cannot be selected).

get_selected_item

Use this method to find out which item is currently selected.

```
CALL METHOD item_model->get_selected_item
          IMPORTING node_key  = node_key
                    item_name = item_name.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Node containing the item that is selected
item_name **TYPE TV_ITMNAME**		Selected item

get_item_selection

Use this method to find out whether item selection is enabled for the tree model instance. When you instantiate the List or Column Tree Model, you specify whether the user should be able to select individual items or whether clicking on an item automatically selects the entire node.

```
CALL METHOD item_model->get_item_selection
          IMPORTING item_selection = item_selection.
```

Parameter and Type	Opt.	Description
item_selection TYPE AS4FLAG		Flag indicating whether item selection is allowed: • 'X': Yes • ' ': No

delete_items

Use this method to delete a specific set of items.

```
CALL METHOD item_model->delete_items
          EXPORTING item_key_table = item_key_table.
```

Parameter and Type	Opt.	Description
item_key_table **TYPE TREEMIKS**		An internal table, each line of which represents an item that you want to delete. The line type of **TREEMIKS** is **TREEMIKEY**, which has the following structure: • **node_key** (type **TM_NODEKEY**) • **item_name** (type **TV_ITMNAME**)

delete_all_items_of_nodes

Use this method to delete all of the items belonging to a specified set of nodes.

```
CALL METHOD item_model->delete_all_items_of_nodes
          EXPORTING node_key_table = node_key_table.
```

Parameter and Type	Opt.	Description
node_key_table **TYPE TREEMNOTAB**		An internal table, each line of which represents a node of the tree model instance whose items you want to delete. The internal table has the line type **TM_NODEKEY**.

item_set_chosen

Use this method to change the selection of an item with the type checkbox.

```
CALL METHOD item_model->item_set_chosen
          EXPORTING node_key  = node_key
                    item_name = item_name
                    chosen    = chosen.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node containing the checkbox
item_name **TYPE TV_ITMNAME**		Name of the checkbox item
chosen **TYPE AS4FLAG**		Parameter containing the new state of the checkbox: • `'X'`: Checkbox is selected • `' '`: Checkbox is not selected

item_set_disabled

Use this method to set the disabled attribute of an item. A disabled item cannot be selected.

```
CALL METHOD item_model->item_set_disabled
          EXPORTING node_key = node_key
                    item_name = item_name
                    disabled = disabled
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node to which the relevant item belongs
item_name **TYPE TV_ITMNAME**		The item whose disabled attribute you want to change
disabled **TYPE AS4FLAG**		New value for the disabled attribute. Possible values: • `'X'`: Item disabled • `' '`: Item not disabled

item_set_editable

Use this method to set the editable attribute of a checkbox item.

```
CALL METHOD item_model->item_set_editable
          EXPORTING node_key  = node_key
                    item_name = item_name
                    editable  = editable.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		The key of the node to which the relevant item belongs
item_name **TYPE** **TV_ITMNAME**		The name of the item whose editable attribute you want to change
editable **TYPE AS4FLAG**		The new value for the editable attribute. Possible values: • **'X'**: Checkbox can be changed • **' '**: Checkbox cannot be changed

item_set_font

Use this method to set the font for an item.

```
CALL METHOD item_model->item_set_font
          EXPORTING node_key = node_key
                    item_name = item_name
                    font = font
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		Key of the node to which the item belongs
item_name **TYPE** **TV_ITMNAME**		Name of the item
font **TYPE I**		Font to be used for the item. Possible values: • **cl_item_tree_model=>item_font_default**: The default font is used. This is a fixed font for a list tree and a proportional font for a column tree. • **cl_item_tree_model=>item_font_fixed**: A fixed font is used • **cl_item_tree_model=>item_font_variable**: A proportional font is used

item_set_hidden

Use this method to set the *hidden* attribute for an item.

```
CALL METHOD item_model->item_set_hidden
          EXPORTING node_key  = node_key
                    item_name = item_name
                    hidden    = hidden.
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		Key of the node to which the item belongs
item_name **TYPE** **TV_ITMNAME**		Name of the item
hidden TYPE AS4FLAG		Flag to indicate whether the item should be hidden. Possible values: • **'X'**: Hidden • ' ': Visible

item_set_style

Use this method to set the style of an item.

```
CALL METHOD item_model->item_set_style
          EXPORTING node_key  = node_key
                    item_name = item_name
                    style     = style
```

Parameter and Type	Opt.	Description
node_key **TYPE** **TM_NODEKEY**		Key of the node to which the item belongs
item_name **TYPE** **TV_ITMNAME**		Name of the item
style **TYPE I**		The style of the item. Possible values: • **cl_item_tree_model=>style_inherited**: The item has the same style as the node to which it belongs. • **cl_item_tree_model=>style_default**: The item has the default text and background colors • **cl_item_tree_model=>style_intensified** • **cl_item_tree_model=>style_inactive** • **cl_item_tree_model=>style_intensified_critical** • **cl_item_tree_model=>style_emphasized_negative** • **cl_item_tree_model=>style_emphasized_positive** • **cl_item_tree_model=>style_emphasized** • Any further constants of the form **cl_item_tree_model=>style_***

item_set_text

Use this method to set the text for an item.

```
CALL METHOD item_model->item_set_text
          EXPORTING node_key =  node_key
                    item_name = item_name
                    text      = text.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node to which the relevant item belongs
item_name **TYPE TV_ITMNAME**		The item whose text you want to change
text **TYPE TM_ITEMTEXT**		The text for the item. **Note**: Although the text is implemented as a string and can therefore be of any length, only the first 100 characters will actually be displayed in the frontend control.

item_get_text

Use this method to retrieve the text of a given item.

```
CALL METHOD item_model->item_get_text
          EXPORTING node_key  = node_key
                    item_name = item_name
          IMPORTING text      = text.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node to which the relevant item belongs
item_name **TYPE TV_ITMNAME**		The name of the item whose text you want to retrieve
text TYPE **TM_ITEMTXT**		The text of the item

item_set_image

Use this method to set an icon for an item.

```
CALL METHOD item_tree->item_set_image
          EXPORTING node_key  = node_key
                    item_name = item_name
                    image     = image.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		Key of the node to which the item belongs
item_name **TYPE TV_ITMNAME**		Name of the item
image **TYPE TV_IMAGE**		Image for the item. Possible values: • `' '`: No image • `'@XY@'`: The SAP icon with the code XY

Methods of Class CL_LIST_TREE_MODEL

constructor

The constructor method is called automatically when you instantiate the class **cl_list_tree_model**. To do this, you must declare a reference variable as follows:

```
DATA list_model TYPE REF TO cl_list_tree_model.
```

You can then create an instance using the **CREATE OBJECT** statement.

```
CREATE OBJECT list_model
      EXPORTING node_selection_mode = node_selection_mode
                hide_selection      = hide_selection
                item_selection      = item_selection
                with_headers        = with_headers
                hierarchy_header    = hierarchy_header
                list_header         = list_header.
```

Parameter and Type	Opt.	Description
node_selection_mode **TYPE I**		Specifies whether or not multiple nodes can be selected simultaneously. Possible values are • **cl_list_tree_model=>node_sel_mode_single** Only one node at a time may be selected • **cl_list_tree_model=>node_sel_mode_multiple** Multiple nodes may be selected
hide_selection **TYPE AS4FLAG**	X	Specifies whether the selection should be hidden. Possible values are • **'X'** - hide selection • **' '** - Show selection
item_selection **TYPE AS4FLAG**		Specifies whether individual items can be selected. Possible values are: • **'X'** - Items may be selected individually • **' '** - The node can only be selected as a whole
with_headers **TYPE AS4FLAG**		Specifies whether the tree should have one header or two. Possible values are: • **'X'** - The control has both a hierarchy header and a list header. Each can be scrolled separately • **' '** - The control only has a hierarchy header. **Note**: You should only use the List Tree Model with headers where it is not practicable to use the Column Tree Model.
hiearachy_header **TYPE TREEMHHDR**		Contains details of the hierarchy header. The parameter has the structure TREEMHHDR [Page 457].
list_header **TYPE TREEMLHDR**		Contains deatils of the list header. The parameter has the structure TREEMLHDR [Page 457].

add_node

Use this method to add a node to the List Tree Model. The node is initially only added to the tree model at the backend. It is automatically transferred to the tree display at the frontend at the end of the next PBO event.

```
CALL METHOD list_model->add_node
      EXPORTING node_key          = node_key
                relative_node_key = relative_node_key
                relationship      = relationship
                isfolder          = isfolder
                hidden            = hidden
                disabled          = disabled
                style             = style
                no_branch         = no_branch
                expander          = expander
                image             = image
                expanded_image    = expanded_image
                drag_drop_id      = drag_drop_id
                last_hitem        = last_hitem
                user_object       = user_object
                items_incomplete  = items_incomplete
                item_table        = item_table.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		The key by which the node is identified in the tree. **This must be unique throughout the tree**. You should only use letters, digits, and the underscore character in node keys.
relative_node_key **TYPE STRING**	X	The key of a node to which the new node is related in position. If the new node is the first or last root node, this parameter must have the value `' '`.

relationship `I`	X	The relationship between the new node and the node specified in `relative_node_key`. Possible values are: - `CL_TREE_MODEL=>RELAT_FIRST_CHILD` Inserts the new node as the first child node of the node specified in `relative_node_key`. This must be a folder. - `CL_TREE_MODEL=>RELAT_LAST_CHILD` Inserts the new node as the last child node of the node specified in `relative_node_key`. This must be a folder. - `CL_TREE_MODEL=>RELAT_PREV_SIBLING` Inserts the new node directly before the related node at the same level. - `CL_TREE_MODEL=>RELAT_NEXT_SIBLING` Inserts the new node directly after the related node at the same level. - `CL_TREE_MODEL=>RELAT_FIRST_SIBLING` Inserts the new node as the first node at the same level as the related node. - `CL_TREE_MODEL=>RELAT_LAST_SIBLING` Inserts the new node as the last node at the same level as the related node. Note: If `relative_node_key` is empty, the new node is inserted as a root node. Where the above values contain the word `FIRST` or `PREV`, it is inserted as the first root node. Where they contain `LAST` or `NEXT`, it is inserted as the last.
isfolder `TYPE AS4FLAG`		Specifies whether the new node should be a folder or a leaf. Possible values: - `'X'`: Node is a folder - `' '`: Node is a leaf
hidden `TYPE AS4FLAG`	X	Specifies whether the node should be hidden (`'X'`) or visible (`' '`). Default is visible.
disabled `TYPE AS4FLAG`	X	Specifies whether the node can be selected (`' '`) or not (`'X'`). The default is **not** disabled. **Note**: If a node is disabled, actions such as double-clicking it have no effect.
style `TYPE AS4FLAG`	X	Sets the colors of the text and the background for the node. The possible values for this field are any static constant `CL_TREE_MODEL=>STYLE_*`. For further details, refer to the definition of `CL_TREE_MODEL` in the Class Builder.
no_branch `TYPE AS4FLAG`	X	Specifies whether connecting lines should be drawn between the nodes (`' '`) or not (`'X'`). The default is for the lines to be drawn.

expander **TYPE AS4FLAG**	X	May only be set for a folder. If you set this attribute, the closed folder always displays a '+' symbol, even if it is empty. When the user clicks on the folder, the event **EXPAND_NO_CHILDREN** is triggered.
image **TYPE C(6)**	X	Specifies the image used for the node. Possible values: • initial: The system uses the default values (leaf symbol for a leaf, closed folder symbol for a folder) • '@**XY**@': An SAP icon with the code **XY**. • '**BNONE**': No image is displayed. The node text begins at the position in which the image would normally be displayed. If you use this value for a node, you should also use it for all of its other same-level nodes.
expanded_image **TYPE C(6)**	X	Specifies the image used for an open folder. The possible values are the same as those listed above for the *image* parameter.
drag_drop_id **TYPE I**	X	Only relevant if you want the node to be drag and drop-enabled. It contains the handle for a drag and drop object.
last_hitem **TYPE STRING**	X	The name of the last item to appear under the hierarchy heading
user_object **TYPE REF TO OBJECT**	X	Can be assigned any reference to an application object
items_incomplete **TYPE AS4FLAG**	X	Flag indicating that the items table is incomplete. In this case, you load the items on demand. For further information, refer to Loading Items on Demand [Page 256].
item_table **TYPE TREEMLITAB**		Table of items for the node with the line type **TREEMLITEM**. For further information, refer to the documentation of the structure in the ABAP Dictionary.

add_nodes

Use this method to add a set of nodes to the List Tree Model. The nodes are initially only added to the tree model on the application server. They are transferred to the tree display at the frontend at the end of the next PBO event.

```
CALL METHOD list_model->add_nodes
      EXPORTING node_table  = node_table.
```

Parameter and Type	Opt.	Description
node_table **TYPE TREEMLNOTA**		Internal table, each line of which represents a node to be added to the List Tree Model. The internal table has the line type TREEMLNODT [Page 458].

update_nodes

Use this method to change the attributes of nodes in the tree model. You cannot use it to change the **RELATKEY** or **RELATSHIP** attributes. If you want to move a node, use the **MOVE_NODE** method.

```
CALL METHOD list_model->update_nodes
      EXPORTING node_table = node_table.
```

Parameter and Type	Opt.	Description
node_table **TYPE TREEMLUNOT**		An internal table in which each line represents one node whose attributes you want to change. You specify the key of the node, and enter a new value for each attribute that you want to change. Each changeable attribute also has a corresponding flag with the name **U_<attribute>**. You must check this flag for each attribute that you change. For example, if you want to change the hidden attribute for a node from **' '** (not hidden) to **'X'** (hidden), you would enter **'X'** in the **HIDDEN** field and **'X'** in the field **U_HIDDEN** (to indicate that the field must be updated). If you want to change all of the changeable attributes for a given node, you should check the **U_ALL** field instead of all of the individual **U_<attribute>** flag fields. The line type of data type **TREEMLUNOT** is made up as follows: • It includes the structure TREEMLNODT [Page 458]. In these fields, you can enter the changed values. • For each changeable value, there is a flag field **u_<attribute>**, which must have the value **'X'** for each attribute you change.

add_items

Use this method to add new items to a node.

```
CALL METHOD list_model->add_items
          EXPORTING item_table = item_table.
```

Parameter and Type	Opt.	Description
item_table **TYPE TREEMLITAC**		Internal table, each line of which represents an item. It has the following structure: • **node_key**: Contains the key of the node to which you want to add the item • Fields of the structure TREEMLITEM [Page 461]: Contains the remaining item attributes

update_items

Use this method to update existing items in the List Tree Model.

```
CALL METHOD list_tree->update_items
          EXPORTING item_table = item_table.
```

Parameter and Type	Opt.	Description
item_table **TYPE** **TREEMLITAD**		An internal table in which each line represents one item whose attributes you want to change. You specify the key of the node, the name of the item, and enter a new value for each attribute that you want to change. Each changeable attribute also has a corresponding flag with the name **U_<attribute>**. You must check this flag for each attribute that you change. For example, if you want to change the hidden attribute for an item from **' '** (not hidden) to **'X'** (hidden), you would enter **'X'** in the **HIDDEN** field and **'X'** in the field **U_HIDDEN** (to indicate that the field must be updated). If you want to change all of the changeable attributes for a given item, you should check the **U_ALL** field instead of all of the individual **U_<attribute>** flag fields. **TREEMLITAD** has the line type **TREEMLITEF**, which is made up as follows: • **node_key**: The key of the node to which the item belongs • The included structure TREEMLITEM [Page 461]. In these fields, you can enter the changed values. • For each changeable value, there is a flag field **u_<attribute>**, which must have the value **'X'** for each attribute you change.

hierarchy_header_set_t_image

Use this method to set a new image for the hierarchy heading.

```
CALL METHOD list_model->hierarchy_header_set_t_image
          EXPORTING t_image = t_image.
```

Parameter and Type	Opt.	Description
t_image `TYPE C(6)`		The image you want to display in the hierarchy heading. Possible values: • `'  '`: No image • `'@XY@'`: An SAP icon with the code XY • `'BNONE'`: No icon. The display position of the heading is then brought forward to start where the image would otherwise have appeared.

hierarchy_header_set_width

Use this method to set the width of the hierarchy heading.

```
CALL METHOD list_model->hierarchy_header_set_width
          EXPORTING width = width.
```

Parameter and Type	Opt.	Description
width **TYPE I**		• The width of the hiearchy heading in characters.

hierarchy_header_set_text

Use this method to set a new text for the hierarchy heading.

```
CALL METHOD list_model->hierarchy_header_set_text
          EXPORTING text = text.
```

Parameter and Type	Opt.	Description
text **TYPE C(132)**		• The new text for the hierarchy heading

hierarchy_header_set_tooltip

Use this method to set a new tooltip for the hierarchy heading. The tooltip is displayed whenever the mouse pointer is positioned over the heading.

```
CALL METHOD list_model->hierarchy_header_set_tooltip
          EXPORTING tooltip = tooltip.
```

Parameter and Type	Opt.	Description
tooltip **TYPE C(132)**		The text for the tooltip

hierarchy_header_adjust_width

Use this method to set a new width for the hierarchy heading. You specify the width in characters.

```
CALL METHOD list_model->hierarchy_header_adjust_width
          EXPORTING width = width
```

Parameter and Type	Opt.	Description
width **TYPE I**		The new width of the hierarchy heading.

hierarchy_header_get_width

Use this method to return the current width of the hierarchy heading.

```
CALL METHOD list_model->hierarchy_header_get_width
          IMPORTING width = width.
```

Parameter and Type	Opt.	Description
width		The width of the hierarchy heading in characters

hierarchy_header_get_props

Use this method to find out the current properties of the hierarchy heading.

```
CALL METHOD list_model->hierarchy_header_get_props
          IMPORTING properties = properties
```

Parameter and Type	Opt.	Description
properties		A structure containing the properties of the hierarchy heading. It has the type TREEMHHDR [Page 457].

list_header_set_t_image

Use this method to set a new image for the list heading.

```
CALL METHOD list_model->list_header_set_t_image
          EXPORTING t_image = t_image.
```

Parameter and Type	Opt.	Description
t_image **TYPE C(6)**		The image you want to display in the list heading. Possible values: • `'  '`: No image • `'@XY@'`: An SAP icon with the code XY • `'BNONE'`: No icon. The display position of the heading is then brought forward to start where the image would otherwise have appeared.

list_header_set_text

Use this method to set a new text for the list heading.

```
CALL METHOD list_model->list_header_set_text
          EXPORTING text = text.
```

Parameter and Type	Opt.	Description
text **TYPE C(132)**		• The new text for the list heading

list_header_set_tooltip

Use this method to set a new tooltip for the list heading. The tooltip is displayed whenever the mouse pointer is positioned over the list heading.

```
CALL METHOD list_model->list_header_set_tooltip
          EXPORTING tooltip = tooltip.
```

Parameter and Type	Opt.	Description
tooltip `TYPE C(132)`		The text for the tooltip

list_header_get_properties

Use this method to return the current properties of the list heading.

```
CALL METHOD list_model->list_header_get_properties
          IMPORTING properties = properties
```

Parameter and Type	Opt.	Description
properties		A structure containing the properties of the list heading. It has the type TREEMLHDR [Page 457].

node_set_last_hierarchy_item

Use this method to specify which item should be the last to appear beneath the hierarchy heading. All subsequent items then appear under the list heading.

```
CALL METHOD list_model->node_set_last_hierarchy_item
          EXPORTING node_key            = node_key
                    last_hierarchy_item = last_hierarchy_item.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		Key of the relevant node
last_hierarchy_item **TYPE C(12)**		Last item of the node to be displayed below the hierarchy item

node_get_properties

Use this method to return the properties of a node.

```
CALL METHOD list_model->node_get_properties
          EXPORTING node_key = node_key
          IMPORTING properties = properties.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		The key of the node whose properties you want to find out
properties **TYPE TREEMLNODT**		A structure containing the properties of the node. This has the type TREEMLNODT [Page 458].

node_get_item

Use this method to find out the attributes of a single item of a node.

```
CALL METHOD list_model->node_get_item
          EXPORTING node_key  = node_key
                    item_name = item_name
          IMPORTING item      = item.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		The key of the relevant node
item_name **TYPE C(12)**		The name of the item whose attributes you want to find out
item **TYPE TREEMLITEM**		A structure containing the attributes of the item. It has the type TREEMLITEM [Page 461].

node_get_items

Use this method to find out the attributes of all of the items of a given node.

```
CALL METHOD list_model->node_get_items
          EXPORTING node_key   = node_key
          IMPORTING item_table = item_table.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		The key of the node whose item information you want to retrieve
item_table **TYPE TREEMLITAB**		An internal table, each line of which contains the attributes of one item of the node. The table has the line type TREEMLITEM [Page 461].

item_set_alignment

Use this method to set the alignment of an item in the List Tree Model.

```
CALL METHOD list_model->item_set_alignment
          EXPORTING node_key = node_key
                    item_name = item_name
                    alignment = alignment.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		Key of the node to which the relevant item belongs
item_name **TYPE C(12)**		Name of the item whose alignment you want to set
alignment **TYPE I**		The alignment of the item. Possible values: • **cl_item_tree_model=>align_left** • **cl_item_tree_model=>align_right** • **cl_item_tree_model=>align_auto** The item is not aligned, but the display width is adjusted to the length of the item

item_set_length

Use this method to set the length of an item in the List Tree Model.

```
CALL METHOD list_model->item_set_length
          EXPORTING node_key = node_key
                    item_name = item_name
                    length = length.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		Key of the node to which the relevant item belongs
item_name **TYPE C(12)**		Name of the item whose length you want to adjust
length **TYPE I**		New length of the item in characters.

get_tree

Use this method to return the contents of the List Tree Model instance in a series of internal tables.

```
CALL METHOD list_model->get_tree
       EXPORTING root_node_key = root_node_key
       IMPORTING node_table    = node_table
                 item_table    = item_table.
```

Parameter and Type	Opt.	Description
root_node_key **TYPE STRING**		The root node of the tree
node_table **TYPE TREEMLNOTA**		Internal table, each line of which represents a node of the List Tree Model instance. The table has the line type TREEMLNODT [Page 458].
item_table **TYPE TREEMLITAC**		Internal table, each line of which represents an item from the List Tree Model instance. It is structured as follows: • node_key: The key of the node to which the item belongs. • The included structure TREEMLITEM [Page 461], which contains the attributes of the items.

set_item_provider

Use this method to specify a reference variable that points to the source for items that are to be loaded on demand [Page 256].

```
CALL METHOD list_model->set_item_provider
          EXPORTING item_provider = item_provider.
```

Parameter and Type	Opt.	Description
item_provider **TYPE REF TO IF_LIST_TREE_MODEL_ITEM_PROV**		Reference variable pointing to the object from which the items are loaded. For further information, refer to Loading Items on Demand [Page 256].

get_with_headers

Use this method to find out if the List Tree Model instance has headings.

```
CALL METHOD list_model->get_with_headers
          IMPORTING with_headers = with_headers.
```

Parameter and Type	Opt.	Description
with_headers **TYPE AS4FLAG**		Indicates whether the List Tree Model instance has headings. Possible values: 'X': Yes ' ': No

Methods of Class CL_COLUMN_TREE_MODEL

constructor

The constructor method is called automatically when you instantiate the class `cl_column_tree_model`. To do this, you must declare a reference variable as follows:

```
DATA column_model TYPE REF TO cl_column_tree_model.
```

You can then create an instance using the **CREATE OBJECT** statement.

```
CREATE OBJECT column_model
      EXPORTING node_selection_mode = node_selection_mode
                hide_selection      = hide_selection
                item_selection      = item_selection
                hierarchy_column_name = hierarchy_colunm_name
                hierarchy_header    = hierarchy_header.
```

Parameter and Type	Opt.	Description
node_selection_mode **TYPE I**		Specifies whether or not multiple nodes can be selected simultaneously. Possible values are • **cl_column_tree_model=>node_sel_mode_single** Only one node at a time may be selected • **cl_column_tree_model=>node_sel_mode_multiple** Multiple nodes may be selected
hide_selection **TYPE AS4FLAG**	X	Specifies whether the selection should be hidden. Possible values: • **'X'**: Selection is hidden • **' '**: Selection is visible
item_selection **TYPE AS4FLAG**	X	Specifies whether items can be selected individually. Possible values: • **'X'**: Items can be selected individually • **' '**: Items cannot be selected individually. Clicking on an item selects the whole node.
hierarchy_column_name **TYPE C(12)**		The name of the column that appears under the hierarchy heading.
hierarchy_header **TYPE TREEMHHDR**		A structure containing information about the hierarchy heading. For full details, refer to Structures for Headings of Item Trees [Page 457].

add_node

Use this method to add a node to the Column Tree Model. Initially, the node is only added to the tree on the application server. It is transferred to the tree display at the frontend at the end of the next PBO event.

```
CALL METHOD column_model->add_node
          EXPORTING node_key = node_key
                    relative_node_key = relative_node_key
                    relationship = relationship
                    isfolder = isfolder
                    hidden = hidden
                    disabled = disabled
                    style = style
                    no_branch = no_branch
                    expander = expander
                    image = image
                    expanded_image = expanded_image
                    drag_drop_id = drag_drop_id
                    user_object = user_object
                    items_incomplete = items_incomplete
                    item_table = item_table.
```

Parameter and Type	Opt.	Description
node_key **TYPE STRING**		The key by which the node is identified in the tree. **This must be unique throughout the tree**. You should only use letters, digits, and the underscore character in node keys.
relative_node_key **TYPE STRING**	X	The key of a node to which the new node is related in position. If the new node is the first or last root node, this parameter must have the value `' '`.

relationship **`TYPE I`**	X	The relationship between the new node and the node specified in **`relative_node_key`**. Possible values are: • **`CL_TREE_MODEL=>RELAT_FIRST_CHILD`** Inserts the new node as the first child node of the node specified in **`relative_node_key`**. This must be a folder. • **`CL_TREE_MODEL=>RELAT_LAST_CHILD`** Inserts the new node as the last child node of the node specified in **`relative_node_key`**. This must be a folder. • **`CL_TREE_MODEL=>RELAT_PREV_SIBLING`** Inserts the new node directly before the related node at the same level. • **`CL_TREE_MODEL=>RELAT_NEXT_SIBLING`** Inserts the new node directly after the related node at the same level. • **`CL_TREE_MODEL=>RELAT_FIRST_SIBLING`** Inserts the new node as the first node at the same level as the related node. • **`CL_TREE_MODEL=>RELAT_LAST_SIBLING`** Inserts the new node as the last node at the same level as the related node. Note: If **`relative_node_key`** is empty, the new node is inserted as a root node. Where the above values contain the word **`FIRST`** or **`PREV`**, it is inserted as the first root node. Where they contain **`LAST`** or **`NEXT`**, it is inserted as the last.
isfolder **`TYPE AS4FLAG`**		Specifies whether the node is a folder. Possible values: • **`'X'`**: Node is a folder • **`' '`**: Node is a leaf
hidden **`TYPE AS4FLAG`**	X	Specifies whether the node is hidden. Possible values: • **`'X'`**: Node is hidden • **`' '`**: Node is visible
disabled **`TYPE AS4FLAG`**	X	Specifies whether the node can be selected (**`' '`**) or not (**`'X'`**). The default is **not** disabled. **Note**: If a node is disabled, actions such as double-clicking it have no effect.
style **`TYPE I`**	X	Sets the colors of the text and the background for the node. The possible values for this field are any static constant **`CL_TREE_MODEL=>STYLE_*`**. For further details, refer to the definition of **`CL_TREE_MODEL`** in the Class Builder.
no_branch **`TYPE AS4FLAG`**	X	Specifies whether connecting lines should be drawn between the nodes (**`' '`**) or not (**`'X'`**). The default is for the lines to be drawn.

expander **TYPE AS4FLAG**	X	May only be set for a folder. If you set this attribute, the closed folder always displays a '+' symbol, even if it is empty. When the user clicks on the folder, the event **EXPAND_NO_CHILDREN** is triggered.
image **TYPE C(6)**	X	Specifies the image used for the node. Possible values: • initial: The system uses the default values (leaf symbol for a leaf, closed folder symbol for a folder) • **'@XY@'**: An SAP icon with the code **XY**. • **'BNONE'**: No image is displayed. The node text begins at the position in which the image would normally be displayed. If you use this value for a node, you should also use it for all of its other same-level nodes.
expanded_image **TYPE C(6)**	X	Specifies the image used for an open folder. The possible values are the same as those listed above for the *image* parameter.
drag_drop_id **TYPE I**	X	Only relevant if you want the node to be drag and drop-enabled. It contains the handle for a drag and drop object.
user_object **TYPE REF TO OBJECT**	X	Can be assigned any reference to an application object
items_incomplete **TYPE AS4FLAG**	X	Flag indicating that the items table is incomplete. In this case, you load the items on demand.
item_table **TYPE TREEMCITAB**		Table containing details of the items of the node. The internal table **TREEMCITAB** has the line type TREEMCITEM [Page 464].

add_nodes

Use this method to add a set of nodes to the Column Tree Model. Initially, the nodes are only added to the tree on the application server. They are transferred to the tree display at the frontend at the end of the next PBO event.

```
CALL METHOD column_model->add_nodes
          EXPORTING node_table = node_table.
```

Parameter and Type	Opt.	Description
node_table **TYPE** **TREEMCNOTA**		Internal table containing the nodes you want to add to the tree. The table has the structure TREEMCNODT [Page 466].

update_nodes

Use this method to change the attributes of one or more nodes in the Column Tree Model.

➡

You cannot use this method to change the **RELATKEY** or **RELATSHIP** attributes of a node. To move a node, use the method MOVE_NODE [Page 312].

```
CALL METHOD column_model->update_nodes
          EXPORTING node_table = node_table.
```

Parameter and Type	Opt.	Description
node_table **TYPE** **TREEMCUNOT**		An internal table in which each line represents one node whose attributes you want to change. You specify the key of the node, and enter a new value for each attribute that you want to change. Each changeable attribute also has a corresponding flag with the name **U_<attribute>**. You must check this flag for each attribute that you change. For example, if you want to change the hidden attribute for a node from **' '** (not hidden) to **'X'** (hidden), you would enter **'X'** in the **HIDDEN** field and **'X'** in the field **U_HIDDEN** (to indicate that the field must be updated). If you want to change all of the changeable attributes for a given node, you should check the **U_ALL** field instead of all of the individual **U_<attribute>** flag fields. The line type of the table is made up as follows: • The included structure TREEMCNODT [Page 466] • For each changeable value, there is a flag field **u_<attribute>**, which must have the value **'X'** for each attribute you change.

add_items

Use this method to add new items to a node in the Column Tree Model.

```
CALL METHOD column_model->add_items
          EXPORTING item_table = item_table.
```

Parameter and Type	Opt.	Description
item_table **TYPE TREEMCITAC**		Internal table, each line of which represents an item. It has the following structure: • **node_key**: Contains the key of the node to which you want to add the item • Fields of the structure TREEMCITEM [Page 464]: Contains the remaining item attributes

update_items

Use this method to change the items of a node in the Column Tree Model.

```
CALL METHOD column_model->update_items
          EXPORTING item_table = item_table.
```

Parameter and Type	Opt.	Description
item_table **TYPE TREEMCITAD**		An internal table in which each line represents one item whose attributes you want to change. You specify the key of the node, the name of the item, and enter a new value for each attribute that you want to change. Each changeable attribute also has a corresponding flag with the name **U_<attribute>**. You must check this flag for each attribute that you change. For example, if you want to change the hidden attribute for an item from ' ' (not hidden) to **'X'** (hidden), you would enter **'X'** in the **HIDDEN** field and **'X'** in the field **U_HIDDEN** (to indicate that the field must be updated). If you want to change all of the changeable attributes for a given item, you should check the **U_ALL** field instead of all of the individual **U_<attribute>** flag fields. **TREEMCITAD** has the line type **TREEMCITEF**, which is made up as follows: • **node_key**: The key of the node to which the item belongs • The included structure TREEMCITEM [Page 464]. In these fields, you can enter the changed values. • For each changeable value, there is a flag field **u_<attribute>**, which must have the value **'X'** for each attribute you change.

add_column

Use this method to add a new column to the Column Tree Model. The column is not inserted under the hierarchy heading.

```
CALL METHOD column_model->add_column
          EXPORTING name           = name
                    hidden         = hidden
                    disabled       = disabled
                    alignment      = alignment
                    width          = width
                    header_image   = header_image
                    header_text    = header_text
                    header_tooltip = header_tooltip.
```

Parameter and Type	Opt.	Description
name **TYPE TV_ITMNAME**		Name of the column
hidden **TYPE AS4FLAG**	X	Flag indicating whether the column is hidden. Possible values: • `'X'`: Column is hidden • `' '`: Column is visible
disabled **TYPE AS4FLAG**	X	Flag indicating whether the column is disabled. Disabled columns cannot be selected. • `'X'`: Column is disabled • `' '`: Column is not disabled
alignment **TYPE I**	X	Alignment of the column. Possible values: • `cl_column_tree_model=>align_left` • `cl_column_tree_model=>align_right` • `cl_column_tree_model=>align_center`
width **TYPE I**		Width of the column in characters
header_image **TYPE TV_IMAGE**	X	Icon to be displayed in the column heading. Possible values: • `' '`: No icon • `'@XY@'`: The SAP icon with code XY
header_text **TYPE TV_HEADING**		The text of the column heading
header_tooltip **TYPE TV_HEADING**	X	Text of the column heading tooltip. This is displayed whenever the mouse pointer is positioned over the heading.

add_hierarchy_column

Use this method to insert a new column within the hierarchy area. The column heading appears below the hierarchy heading.

```
CALL METHOD column_model->add_hierarchy_column
          EXPORTING name     = name
                    hidden   = hidden
                    disabled = disabled.
```

Parameter and Type	Opt.	Description
name **TYPE** **TV_ITMNAME**		Name of the column
hidden **TYPE AS4FLAG**	X	Indicates whether the column should be hidden (**'X'**) or visible (' ')
disabled **TYPE AS4FLAG**	X	Indicates whether the column should be disabled (**'X'**) or enabled (' ')

insert_column

Use this method to insert a new column in the Column Tree Model after an existing column.

If you want to add a column to the end of the Column Tree Model, use the add_column [Page 408] method.

```
CALL METHOD column_model->insert_column
          EXPORTING name                   = name
                    predecessor_column = predecessor_column
                    hidden                 = hidden
                    disabled               = disabled
                    alignment              = alignment
                    width                  = width
                    header_image           = header_image
                    header_text            = header_text
                    header_tooltip         = header_tooltip.
```

Parameter and Type	Opt.	Description
name **TYPE TV_ITMNAME**		Name of the column
predecessor_column **TYPE TV_ITMNAME**	X	The column after which you want to insert the new column
hidden **TYPE AS4FLAG**	X	Flag indicating whether the column is hidden. Possible values: • **'X'**: Column is hidden • **' '**: Column is visible
disabled **TYPE AS4FLAG**	X	Flag indicating whether the column is disabled. Disabled columns cannot be selected. • **'X'**: Column is disabled • **' '**: Column is not disabled
alignment **TYPE I**	X	Alignment of the column. Possible values: • **cl_column_tree_model=>align_left** • **cl_column_tree_model=>align_right** • **cl_column_tree_model=>align_center**
width **TYPE I**		Width of the column in characters

header_image **TYPE TV_IMAGE**	X	Icon to be displayed in the column heading. Possible values: • `' '`: No icon • `'@XY@'`: The SAP icon with code XY
header_text **TYPE** **TV_HEADING**		The text of the column heading
header_tooltip **TYPE** **TV_HEADING**	X	Text of the column heading tooltip. This is displayed whenever the mouse pointer is positioned over the heading.

insert_hierarchy_column

Use this method to insert a column at a given position under the hierarchy header. If you want to add a hierarchy column at the right-hand end, use the add_hierarchy_column [Page 410] method.

```
CALL METHOD column_model->insert_hierarchy_column
          EXPORTING name     = name
                    hidden   = hidden
                    disabled = disabled.
```

Parameter and Type	Opt.	Description
name **TYPE TV_ITMNAME**		Name of the column
hidden **TYPE AS4FLAG**	X	Indicates whether the column should be hidden (`'X'`) or visible (`' '`)
disabled **TYPE AS4FLAG**	X	Indicates whether the column should be disabled (`'X'`) or enabled (`' '`)
predecessor_column **TYPE TV_ITMNAME**	X	Name of the column after which you want to insert the new column

delete_column

Use this method to delete a column from the Column Tree Model.

```
CALL METHOD column_model->delete_column
          EXPORTING column_name = column_name.
```

Parameter and Type	Opt.	Description
column_name **TYPE TV_ITMNAME**		The name of the column you want to delete

hierarchy_header_adjust_width

Use this method to adjust the width of the hierarchy header in the Column Tree Model so that all of the items below it are fully visible. Only expanded nodes are taken into account.

```
CALL METHOD column_model->hierarchy_header_adjust_width
          EXPORTING include_heading = include_heading.
```

Parameter and Type	Opt.	Description
include_heading **TYPE AS4FLAG**	X	Specifies whether the heading should be included in the calculation for the width adjustment (`'X'`) or not (`' '`).

hierarchy_header_set_text

Use this method to set a new text for the hierarchy heading.

```
CALL METHOD column_model->hierarchy_header_set_text
          EXPORTING text = text.
```

Parameter and Type	Opt.	Description
text TYPE TV_HEADING		New heading text

hierarchy_header_set_tooltip

Use this method to set a new tooltip for the hierarchy heading. The tooltip is displayed whenever the mouse pointer is positioned over the hierarchy heading.

```
CALL METHOD column_model->hierarchy_header_set_tooltip
          EXPORTING toltip = tooltip.
```

Parameter and Type	Opt.	Description
tooltip **TYPE TV_HEADING**	X	The new tooltip text

hierarchy_header_set_t_image

Use this method to set a new icon for the hierarchy heading.

```
CALL METHOD column_model->set_t_image
          EXPORTING .
```

Parameter and Type	Opt.	Description
t_image **TYPE** **TV_IMAGE**		The new icon for the hierarchy heading in the form `'@XY@'`. For no icon, use the value `'  '`.

hierarchy_header_set_width

Use this method to set a new width for the hierarchy heading.

```
CALL METHOD column_model->hierarchy_header_set_width
          EXPORTING width = width.
```

Parameter and Type	Opt.	Description
width **TYPE I**		The new width of the hierarchy heading in characters

update_hierarchy_header

Use this method to change the attributes of the hierarchy header. There are four attributes that you can change using this method. For each attribute that you change, you must also set the corresponding **UPDATE_<attribute name>** parameter to **'X'**.

```
CALL METHOD column_model->update_hierarchy_header
          EXPORTING t_image = t_image
                    width   = width
                    heading = heading
                    tooltip = tooltip
                    update_t_image = update_t_image
                    update_width   = update_width
                    update_heading = update_heading
                    update_tooltip = update_tooltip.
```

Parameter and Type	Opt.	Description
t_image **TYPE** **TV_IMAGE**	X	A new icon for the hierarchy heading in the form **'@XY@'**. For no icon, use the value **' '**.
width **TYPE I**	X	The new width of the hierarchy heading
heading **TYPE** **TV_HEADING**	X	The new heading text for the hierarchy heading
tooltip **TYPE** **TV_HEADING**	X	The new tooltip for the hierarchy heading. The tooltip is displayed whenever the mouse pointer is positioned over the hierarchy heading.
update_t_image **TYPE AS4FLAG**	X	**'X'** if you entered a new value for **t_image**
update_width **TYPE AS4FLAG**	X	**'X'** if you entered a new value for **width**
update_heading **TYPE AS4FLAG**	X	**'X'** if you entered a new value for **heading**
update_tooltip **TYPE AS4FLAG**	X	**'X'** if you entered a new value for **tooltip**

hierarchy_header_get_width

Use this method to return the width in characters of the hierarchy heading.

```
CALL METHOD column_model->hierarchy_header_get_width
          IMPORTING width = width.
```

Parameter and Type	Opt.	Description
width **TYPE I**		Width of the hierarchy heading

hierarchy_header_get_props

Use this method to return the current properties of the hierarchy heading.

```
CALL METHOD column_model->hierarchy_header_get_props
          IMPORTING properties = properties.
```

Parameter and Type	Opt.	Description
properties **TYPE TREEMHHDR**		A structure containing the current properties of the hierarchy heading. For further information, refer to Structures for Headings of Item Trees [Page 457].

get_hierarchy_columns

Use this method to return the names of the columns under the hierarchy heading.

```
CALL METHOD column_model->get_hierarchy_columns
          IMPORTING column_table = column_table.
```

Parameter and Type	Opt.	Description
column_table **TYPE TREEMCHCLS**		Table containing the columns under the hierarchy heading. It has the line type **TREEMCHCL**.

Structure TREEMCHCL

Component and Type	Description
name **TYPE TV_ITMNAME**	The name of the column
hidden **TYPE AS4FLAG**	Flag indicating whether the column is hidden (**'X'**) or not (**' '**)
disabled **TYPE AS4FLAG**	Flag indicating whether the column is disabled (**'X'**) or not (**' '**)

get_nr_of_columns

Use this method to find out the number of columns in the Column Tree Model.

```
CALL METHOD column_model->get_nr_of_columns
          IMPORTING nr_of_columns = nr_of_columns.
```

Parameter and Type	Opt.	Description
nr_of_columns **TYPE I**		The number of columns in the tree model

get_first_column

Use this method to find out the name of the first column in the Column Tree Model instance.

```
CALL METHOD column_model->get_first_column
          IMPORTING column_name = column_name.
```

Parameter and Type	Opt.	Description
column_name **TYPE TV_ITMNAME**		The name of the first column in the tree model instance

get_last_column

Use this method to find out the name of the last column in the Column Tree Model instance.

```
CALL METHOD column_model->get_last_column
          IMPORTING column_name = column_name.
```

Parameter and Type	Opt.	Description
column_name **TYPE TV_ITMNAME**		The name of the last column in the tree model instance

get_widths_of_columns

Use this mehtod to find out the widths of all of the columns in the Column Tree Model instance.

```
CALL METHOD column_model->get_widths_of_columns
          IMPORTING widths_of_columns = widths_of_columns.
```

Parameter and Type	Opt.	Description
widths_of_columns **TYPE TREEV_COWT**		An internal table containing the widths of the columns. It has the line type **TREEV_COWI**.

Structure TREEV_COWI

Component and Type	Description
name **TYPE TV_ITMNAME**	Name of the column
width_pix **TYPE I**	Width of the column in pixels
width_char **TYPE I**	Width of the column in characters

get_column_order

Use this method to find out the order of the columns in the Column Tree Model instance.

```
CALL METHOD column_model->get_column_order
          IMPORTING columns = columns.
```

Parameter and Type	Opt.	Description
columns **TYPE TREEV_CONA**		An internal table in which each line contains the name of a column. The order in which they are listed is their order in the tree model. The table has the line type **TV_ITMNAME**.

set_column_order

Use this method to set the order of the columns in the Column Tree Model.

```
CALL METHOD column_model->set_column_order
          EXPORTING columns = columns.
```

Parameter and Type	Opt.	Description
columns **TYPE TREEV_CONA**		An internal table with line type TV_ITMNAME, each line of which should contain the name of a column. The columns will appear in the order in which you list them in the table.

set_column_order_frozen

In a column tree, the user can swap the positions of columns using the mouse. Use this method to disable and enable this feature.

```
CALL METHOD column_model->set_column_order_frozen
          EXPORTING frozen = frozen.
```

Parameter and Type	Opt.	Description
frozen **TYPE AS4FLAG**		Specifies whether the column order is frozen. Possible values: • `'X'`: Column order is frozen (cannot be changed by the user) • `' '`: Column order is not frozen (can be changed by the user)

column_set_disabled

Use this method to set the *disabled* attribute of a column in the tree model. A disabled column cannot be selected.

```
CALL METHOD column_model->column_set_disabled
          EXPORTING column_name = column_name
                    disabled    = disabled.
```

Parameter and Type	Opt.	Description
column_name **TYPE TV_ITMNAME**		The name of the column in the tree model instance
disabled **TYPE AS4FLAG**		Flag indicating whether the column should be disabled. Possible values: • **'X'**: Column is disabled • **' '**: Column is not disabled

column_set_heading_image

Use this method to set a new icon for a column heading.

```
CALL METHOD column_model->column_set_heading_image
          EXPORTING .
```

Parameter and Type	Opt.	Description
column_name **TYPE** **TV_ITMNAME**		The name of the column in the tree model instance
image **TYPE** **TV_IMAGE**		The image you want to display in the column heading. Possible values: • `'@XY@'`: The SAP icon with the code XY • `'  '`: No image

column_set_heading_text

Use this method to set a new text for a column heading in the Column Tree Model.

```
CALL METHOD column_model->column_set_heading_text
          EXPORTING column_name = column_name
                    text        = text.
```

Parameter and Type	Opt.	Description
column_name **TYPE TV_ITMNAME**		The name of the column in the tree model instance
text **TYPE TV_HEADING**		The new text for the column heading

column_set_heading_tooltip

Use this method to set a new tooltip for a column heading. The tooltip is displayed whenever the mouse pointer is positioned over the heading.

```
CALL METHOD column_model->column_set_heading_tooltip
          EXPORTING column_name = column_name
                    tooltip     = tooltip.
```

Parameter and Type	Opt.	Description
column_name **TYPE TV_ITMNAME**		The name of the column in the tree model instance
tooltip **TYPE TV_HEADING**		The text for the new tooltip

column_set_hidden

Use this method to hide a column in the Column Tree Model. You also use it to make a hidden column visible again.

```
CALL METHOD column_model->column_set_hidden
          EXPORTING column_name = column_name
                    hidden      = hidden.
```

Parameter and Type	Opt.	Description
column_name **TYPE TV_ITMNAME**		The name of the column in the tree model instance
hidden **TYPE AS4FLAG**		Flag to indicate whether or not the column is hidden. Possible values: • `'X'`: Hidden • `' '`: Visible

column_set_width

Use this method to set the width of a column in the Column Tree Model.

```
CALL METHOD column_model->column_set_width
          EXPORTING column_name = column_name
                    width       = width.
```

Parameter and Type	Opt.	Description
column_name **TYPE TV_ITMNAME**		The name of the column in the tree model instance
width **TYPE I**		New width of the column in characters

update_column

Use this method to change the attributes of a column header. There are four attributes that you can change using this method. For each attribute that you change, you must also set the corresponding **UPDATE_<attribute name>** parameter to **'X'**.

```
CALL METHOD column_model->update_column
          EXPORTING name                  = name
                    hidden                = hidden
                    disabled              = disabled
                    alignment             = alignment
                    header_image          = header_image
                    header_text           = header_text
                    header_tooltip        = header_tooltip
                    width                 = width
                    update_hidden         = update_hidden
                    update_disabled       = update_disabled
                    update_alignment      = update_alignment
                    update_header_image   = update_header_image
                    update_header_text    = update_header_text
                    update_header_tooltip = update_header_tooltip
                    update_width          = update_width.
```

Parameter and Type	Opt.	Description
name **TYPE TV_ITMNAME**		Name of the column
hidden **TYPE AS4FLAG**	X	Flag indicating whether the column is hidden. Possible values: • **'X'**: Column is hidden • **' '**: Column is visible
disabled **TYPE AS4FLAG**	X	Flag indicating whether the column is disabled. Disabled columns cannot be selected. • **'X'**: Column is disabled • **' '**: Column is not disabled
alignment **TYPE I**	X	Alignment of the column. Possible values: • **cl_column_tree_model=>align_left** • **cl_column_tree_model=>align_right** • **cl_column_tree_model=>align_center**
header_image **TYPE TV_IMAGE**	X	Icon to be displayed in the column heading. Possible values: • **' '**: No icon • **'@XY@'**: The SAP icon with code XY

header_text **TYPE TV_HEADING**	X	The text of the column heading
header_tooltip **TYPE TV_HEADING**	X	Text of the column heading tooltip. This is displayed whenever the mouse pointer is positioned over the heading.
width **TYPE I**		Width of the column in characters
update_hidden **TYPE AS4FLAG**	X	'X' if you entered a new value in **hidden**
update_disabled **TYPE AS4FLAG**	X	'X' if you entered a new value in **disabled**
update_alignment **TYPE AS4FLAG**	X	'X' if you entered a new value in **alignment**
update_header_image **TYPE AS4FLAG**	X	'X' if you entered a new value in **header_image**
update_header_text **TYPE AS4FLAG**	X	'X' if you entered a new value in **header_text**
update_header_tooltip **TYPE AS4FLAG**	X	'X' if you entered a new value in **header_tooltip**
update_width **TYPE AS4FLAG**	X	'X' if you entered a new value in **width**

adjust_column_width

Use this method to adjust the widths of a selected range of columns so that their entire contents are visible. The method only takes into account nodes that are already expanded.

```
CALL METHOD column_model->adjust_column_width
          EXPORTING start_column    = start_column
                    end_column      = end_column
                    all_columns     = all_columns
                    include_heading = include_heading.
```

Parameter and Type	Opt.	Description
start_column **TYPE TV_ITMNAME**	X	The name of the first column in the range
end_column **TYPE TV_ITMNAME**	X	The name of the last column in the range
all_columns **TYPE AS4FLAG**	X	Flag: Adjust the width of all columns, taking into account the column headings as well (**all_columns = 'X'**)
include_heading **TYPE AS4FLAG**	X	Flag: Indicates whether the column headings should be taking into account when calculating the required width (**include_heading = 'X'**)

column_get_width

Use this method to find out the width of a particular column.

```
CALL METHOD column_model->column_get_width
          EXPORTING column = column
          IMPORTING width = width.
```

Parameter and Type	Opt.	Description
column **TYPE TV_ITMNAME**		Name of the column
width **TYPE I**		Width of the column in characters

column_get_next_sibling

Use this method to find out the name of the next same-level column in the tree model.

```
CALL METHOD column_model->column_get_next_sibling
          EXPORTING column              = column
          IMPORTING sibling_column_name = sibling_column_name.
```

Parameter and Type	Opt.	Description
column **TYPE TV_ITMNAME**		Name of the column
sibling_column_name **TYPE TV_ITMNAME**		Name of the next column.

column_get_prev_sibling

Use this method to find out the name of the column preceding any given column in the Column Tree Model.

```
CALL METHOD column_model->column_get_prev_sibling
          EXPORTING column              = column
          IMPORTING sibling_column_name = sibling_column_name.
```

Parameter and Type	Opt.	Description
column **TYPE TV_ITMNAME**		Name of the column
sibling_column_name **TYPE TV_ITMNAME**		Name of the previous column.

column_get_properties

Use this method to find out the properties of a given column.

```
CALL METHOD column_model->column_get_properties
          EXPORTING column     = column
          IMPORTING properties = properties.
```

Parameter and Type	Opt.	Description
column **TYPE TV_ITMNAME**		Name of the column
properties **TYPE TREEMCCOL**		A structure containing various attributes of the column

node_get_item

Use this method to return the properties of an item of a node.

```
CALL METHOD column_model->node_get_item
          EXPORTING node_key  = node_key
                    item_name = item_name
          IMPORTING item      = item.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node to which the item belongs
item_name **TYPE TV_ITMNAME**		The item whose properties you want to find out
item **TYPE TREEMCITEM**		A structure containing the properties of the node. For further information, refer to Structure TREEMCITEM [Page 464]

➡

If you want to find out the properties of all of the items of a given node, use the method node_get_items [Page 445].

node_get_items

Use this method to find out the properties of all of the items belonging to a given node in the Column Tree Model.

```
CALL METHOD column_model->node_get_items
          EXPORTING node_key   = node_key
          IMPORTING item_table = item_table.
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the relevant node
item_table **TYPE TREEMCITAB**		An internal table, each line of which represents one item of the node specified in **node_key**. The internal table has the line type TREEMCITEM [Page 464].

node_get_properties

Use this method to find out the properties of a node in the Column Tree Model.

```
CALL METHOD column_model->node_get_properties
          EXPORTING node_key   = node_key
          IMPORTING properties = properties
```

Parameter and Type	Opt.	Description
node_key **TYPE TM_NODEKEY**		The key of the node whose properties you want to find out
properties **TYPE TREEMSNOD**		A structure containing the properties of the node. For further information, refer to Structure TREEMSNOD [Page 469].

get_table

Use this method to return the contents of the Column Tree Model instance in a series of internal tables.

```
CALL METHOD column_model->get_tree
       EXPORTING root_node_key = root_node_key
       IMPORTING node_table    = node_table
                 item_table    = item_table.
```

Parameter and Type	Opt.	Description
root_node_key **TYPE STRING**		The root node of the tree
node_table **TYPE** **TREEMCNOTA**		Internal table, each line of which represents a node of the List Tree Model instance. The table has the line type TREEMCNODT [Page 466].
item_table **TYPE** **TREEMCITAC**		Internal table, each line of which represents an item from the List Tree Model instance. It is structured as follows: • **node_key**: The key of the node to which the item belongs. • The included structure TREEMCITEM [Page 464], which contains the attributes of the items.

set_print_short_header_width

Use this method when you want to print the Column Tree Model to set the cut-off point for headings in the print output.

If the width required to print the tree is greater than the value you specify in the width parameter of this method, the width of the headings will not be included in the calculation for the overall width required.

```
CALL METHOD column_model->set_print_short_header_width
          EXPORTING width = width.
```

Parameter and Type	Opt.	Description
width **TYPE I**		Width beyond which the headings are disregarded in the calculation of the overall width required to print the trees

set_item_provider

Use this method to specify a reference variable that points to the source for items that are to be loaded on demand [Page 256].

```
CALL METHOD column_model->set_item_provider
          EXPORTING item_provider = item_provider.
```

Parameter and Type	Opt.	Description
item_provider **TYPE REF TO IF_COLUMN_TREE_MODEL_ITEM_PROV**		Reference variable pointing to the object from which the items are loaded. For further information, refer to Loading Items on Demand [Page 256]

Important Data Structures

Structure TREEMSNODT

Definition

TREEMSNODT is a data structure that describes the attributes of a single node in a Simple Tree Model. Its definition is stored centrally in the ABAP Dictionary, and you can use it to define the data types of your own parameters.

TREEMSNODT is also the line type of the internal table type **TREEMSNOTA.**

Use

You can use **TREEMSNODT** to type the actual parameter **properties** of the method **node_get_properties** in class **cl_simple_tree_model** and to type a work area for internal tables with the type **TREEMSNOTA** (methods **add_nodes** and **get_tree** of the same class).

Structure

Component	Type	Description
node_key	**TM_NODEKEY**	The key by which the node is identified in the tree. **This must be unique thorughout the tree**. You should only use letters, digits, and the underscore character in node keys.
relatkey	**TM_NODEKEY**	The key of a node to which the new node is related in position. If the new node is the first or last root node, this parameter must have the value `' '`.

relatship	`I`	The relationship between the new node and the node specified in `relatkey`. Possible values are: • `CL_TREE_MODEL=>RELAT_FIRST_CHILD` Inserts the new node as the first child node of the node specified in `relative_node_key`. This must be a folder. • `CL_TREE_MODEL=>RELAT_LAST_CHILD` Inserts the new node as the last child node of the node specified in `relative_node_key`. This must be a folder. • `CL_TREE_MODEL=>RELAT_PREV_SIBLING` Inserts the new node directly before the related node at the same level. • `CL_TREE_MODEL=>RELAT_NEXT_SIBLING` Inserts the new node directly after the related node at the same level. • `CL_TREE_MODEL=>RELAT_FIRST_SIBLING` Inserts the new node as the first node at the same level as the related node. • `CL_TREE_MODEL=>RELAT_LAST_SIBLING` Inserts the new node as the last node at the same level as the related node. Note: If `relatkey` is empty, the new node is inserted as a root node. Where the above values contain the word `FIRST` or `PREV`, it is inserted as the first root node. Where they contain `LAST` or `NEXT`, it is inserted as the last.
hidden	`AS4FLAG`	Specifies whether the node should be hidden (`'X'`) or visible (`' '`). Default is visible.
disabled	`AS4FLAG`	Specifies whether the node can be selected (`' '`) or not (`'X'`). The default is **not** disabled. **Note**: If a node is disabled, actions such as double-clicking it have no effect.
isfolder	`AS4FLAG`	Specifies whether the new node should be a folder or a leaf. Possible values: • `'X'`: Node is a folder `' '`: Node is a leaf

n_image	**TV_IMAGE**	Specifies the image used for the node. Possible values: - initial: The system uses the default values (leaf symbol for a leaf, closed folder symbol for a folder) - **'@XY@'**: An SAP icon with the code **XY**. - **'BNONE'**: No image is displayed. The node text begins at the position in which the image would normally be displayed. If you use this value for a node, you should also use it for all of its other same-level nodes.
exp_image	**TV_IMAGE**	Specifies the image used for an open folder. The possible values are the same as those listed above for the *image* parameter.
style	**I**	Sets the colors of the text and the background for the node. The possible values for this field are any static constant **CL_TREE_MODEL=>STYLE_***. For further details, refer to the definition of **CL_TREE_MODEL** in the Class Builder.
no_branch	**AS4FLAG**	Specifies whether connecting lines should be drawn between the nodes (**' '**) or not (**'X'**). The default is for the lines to be drawn.
expander	**AS4FLAG**	May only be set for a folder. If you set this attribute, the closed folder always displays a '+' symbol, even if it is empty. When the user clicks on the folder, the event **EXPAND_NO_CHILDREN** is triggered.
dragdropid	**I**	Only relevant if you want the node to be drag and drop-enabled. It contains the handle for a drag and drop object.
userobject	**REF TO OBJECT**	Can be assigned any reference to an application object
text	**TM_NODETXT**	Text of a node

Structure TREEMSUNO

Definition

TREEMSUNO is a data structure that you use to pass changes in node attributes to the simple tree model. Its definition is stored centrally in the ABAP Dictionary, and you can use it to define the data types of your own parameters.

TREEMSUNO is also the line type of the internal table type **TREEMSUNOT**.

Use

You use this structure to specify the line type of the actual parameter **NODE_TABLE** in the method **UPDATE_NODES** of **CL_SIMPLE_TREE_MODEL**. You can also use it to create a work area for the internal table.

In the structure, you specify the key of the node you want to change, and the new values of each of the relevant changeable attributes. Each attribute also has a corresponding flag field in the structure with the name **U_<attribute>**. You must check this corresponding flag for each attribute that you want to change.

Structure

Component	Type	Description
node_key	**STRING**	The key by which the node is identified in the tree. **This must be unique thorughout the tree**. You should only use letters, digits, and the underscore character in node keys.
relatkey	**STRING**	The key of a node to which the new node is related in position. If the new node is the first or last root node, this parameter must have the value `' '`.

relatship	**`I`**	The relationship between the new node and the node specified in **`relatkey`**. Possible values are: • **`CL_TREE_MODEL=>RELAT_FIRST_CHILD`** Inserts the new node as the first child node of the node specified in **`relative_node_key`**. This must be a folder. • **`CL_TREE_MODEL=>RELAT_LAST_CHILD`** Inserts the new node as the last child node of the node specified in **`relative_node_key`**. This must be a folder. • **`CL_TREE_MODEL=>RELAT_PREV_SIBLING`** Inserts the new node directly before the related node at the same level. • **`CL_TREE_MODEL=>RELAT_NEXT_SIBLING`** Inserts the new node directly after the related node at the same level. • **`CL_TREE_MODEL=>RELAT_FIRST_SIBLING`** Inserts the new node as the first node at the same level as the related node. • **`CL_TREE_MODEL=>RELAT_LAST_SIBLING`** Inserts the new node as the last node at the same level as the related node. Note: If **`relatkey`** is empty, the new node is inserted as a root node. Where the above values contain the word **`FIRST`** or **`PREV`**, it is inserted as the first root node. Where they contain **`LAST`** or **`NEXT`**, it is inserted as the last.
hidden	**`AS4FLAG`**	Specifies whether the node should be hidden (**`'X'`**) or visible (`' '`).
disabled	**`AS4FLAG`**	Specifies whether the node can be selected (`' '`) or not (**`'X'`**).
isfolder	**`AS4FLAG`**	Specifies whether the new node should be a folder or a leaf. Possible values: • **`'X'`**: Node is a folder `' '`: Node is a leaf
n_image	**`C(6)`**	Specifies the image used for the node. Possible values: • initial: The system uses the default values (leaf symbol for a leaf, closed folder symbol for a folder) • **`'@XY@'`**: An SAP icon with the code **`XY`**. • **`'BNONE'`**: No image is displayed. The node text begins at the position in which the image would normally be displayed. If you use this value for a node, you should also use it for all of its other same-level nodes.

exp_image	`C(6)`	Specifies the image used for an open folder. The possible values are the same as those listed above for the *image* parameter.
style	`I`	Sets the colors of the text and the background for the node. The possible values for this field are any static constant `CL_TREE_MODEL=>STYLE_*`. For further details, refer to the definition of `CL_TREE_MODEL` in the Class Builder.
no_branch	`AS4FLAG`	Specifies whether connecting lines should be drawn between the nodes (`' '`) or not (`'X'`). The default is for the lines to be drawn.
expander	`AS4FLAG`	May only be set for a folder. If you set this attribute, the closed folder always displays a '+' symbol, even if it is empty.
dragdropid	`I`	Only relevant if you want the node to be drag and drop-enabled. It contains the handle for a drag and drop object.
userobject	`REF TO OBJECT`	Can be assigned any reference to an application object
text	`STRING`	Node text
u_all	`AS4FLAG`	Indicates that all changeable attributes have been modified
u_hidden	`AS4FLAG`	Indicates that the corresponding attribute has been modified
u_disabled	`AS4FLAG`	
u_isfolder	`AS4FLAG`	
u_n_image	`AS4FLAG`	
u_exp_image	`AS4FLAG`	
u_style	`AS4FLAG`	
u_no_branch	`AS4FLAG`	
u_expander	`AS4FLAG`	
u_dragdropid	`AS4FLAG`	
u_userobject	`AS4FLAG`	
u_text	`AS4FLAG`	

Structures for Headings of Item Trees

Definition

The two structures TREEMHHDR and TREEMLHDR are used to define headings in the List Tree Model and Column Tree Model.

Use

The structures are used as follows:

Structure	Defines	List	Column
TREEMHHDR	A hierarchy heading	✓	✓
TREEMLHDR	A list heading	✓	

Structure

TREEMHDR

Component	Description
t_image **TYPE C(6)**	Icon or image to be used in the heading
heading **TYPE C(132)**	Text of the heading
tooltip **TYPE C(132)**	Text that is displayed when the mouse pointer is positioned over the heading
width **TYPE I**	Width of the heading

TREEMLHDR

Component	Description
t_image **TYPE C(6)**	Icon or image to be used in the heading
heading **TYPE C(132)**	Text of the heading
tooltip **TYPE C(132)**	Text that is displayed when the mouse pointer is positioned over the heading

Structure TREEMLNODT

Definition

TREEMLNODT is a data structure that describes the attributes of a single node in a List Tree Model. Its definition is stored centrally in the ABAP Dictionary, and you can use it to define the data types of your own parameters.

TREEMLNODT is also the line type of the internal table type **TREEMLNOTA.**

Use

You can use **TREEMLNODT** to specify the type of a work area for the actual parameter **node_table** in method **add_nodes** of class **cl_list_tree_model**.

Structure

Component	Type	Description
node_key	STRING	The key by which the node is identified in the tree. **This must be unique thorughout the tree**. You should only use letters, digits, and the underscore character in node keys.
relatkey	STRING	The key of a node to which the new node is related in position. If the new node is the first or last root node, this parameter must have the value `' '`.

relatship	**`I`**	The relationship between the new node and the node specified in **`relatkey`**. Possible values are: • **`CL_TREE_MODEL=>RELAT_FIRST_CHILD`** Inserts the new node as the first child node of the node specified in **`relative_node_key`**. This must be a folder. • **`CL_TREE_MODEL=>RELAT_LAST_CHILD`** Inserts the new node as the last child node of the node specified in **`relative_node_key`**. This must be a folder. • **`CL_TREE_MODEL=>RELAT_PREV_SIBLING`** Inserts the new node directly before the related node at the same level. • **`CL_TREE_MODEL=>RELAT_NEXT_SIBLING`** Inserts the new node directly after the related node at the same level. • **`CL_TREE_MODEL=>RELAT_FIRST_SIBLING`** Inserts the new node as the first node at the same level as the related node. • **`CL_TREE_MODEL=>RELAT_LAST_SIBLING`** Inserts the new node as the last node at the same level as the related node. Note: If **`relatkey`** is empty, the new node is inserted as a root node. Where the above values contain the word **`FIRST`** or **`PREV`**, it is inserted as the first root node. Where they contain **`LAST`** or **`NEXT`**, it is inserted as the last.
hidden	**`AS4FLAG`**	Specifies whether the node should be hidden (**`'X'`**) or visible (`' '`). Default is visible.
disabled	**`AS4FLAG`**	Specifies whether the node can be selected (`' '`) or not (**`'X'`**). The default is **not** disabled. **Note**: If a node is disabled, actions such as double-clicking it have no effect.
isfolder	**`AS4FLAG`**	Specifies whether the new node should be a folder or a leaf. Possible values: • **`'X'`**: Node is a folder `' '`: Node is a leaf

n_image	`C(6)`	Specifies the image used for the node. Possible values: • initial: The system uses the default values (leaf symbol for a leaf, closed folder symbol for a folder) • `'@XY@'`: An SAP icon with the code **XY**. • `'BNONE'`: No image is displayed. The node text begins at the position in which the image would normally be displayed. If you use this value for a node, you should also use it for all of its other same-level nodes.
exp_image	`C(6)`	Specifies the image used for an open folder. The possible values are the same as those listed above for the *image* parameter.
style	`I`	Sets the colors of the text and the background for the node. The possible values for this field are any static constant **CL_TREE_MODEL=>STYLE_***. For further details, refer to the definition of **CL_TREE_MODEL** in the Class Builder.
no_branch	**AS4FLAG**	Specifies whether connecting lines should be drawn between the nodes (`' '`) or not (`'X'`). The default is for the lines to be drawn.
expander	**AS4FLAG**	May only be set for a folder. If you set this attribute, the closed folder always displays a '+' symbol, even if it is empty. When the user clicks on the folder, the event **EXPAND_NO_CHILDREN** is triggered.
dragdropid	`I`	Only relevant if you want the node to be drag and drop-enabled. It contains the handle for a drag and drop object.
userobject	**REF TO OBJECT**	Can be assigned any reference to an application object
itemsincom	**AS4FLAG**	Indicates that the item specification is incomplete. For further information, refer to Loading Items on Demand [Page 256].
last_hitem	`C(12)`	The last item to appear under the hierarchy heading in the tree display

Integration

Structure TREEMLITEM

Definition

TREEMLITEM is a structure that is used to define the line type of the internal table **TREEMLITAB**. This internal table is used in the List Tree Model to specify the items that belong to a particular node.

Use

You can use **TREEMLITAB** to specify the type of the actual parameter you are going to pass to the **items_table** parameter in the **add_node** method of **cl_list_tree_model**.

Structure

Component and Type	**Description**
item_name `TYPE C(12)`	Name of the item
class **TYPE I**	Class of the item. Possible values: • **cl_item_tree_model=>item_class_text**: Item is a text • **cl_item_tree_model=>item_class_button**: Item is a pushbutton • **cl_item_tree_model=>item_class_checkbox**: Item is a checkbox • **cl_item_tree_model=>item_class_link**: Item is a link
font **TYPE I**	Font in which the item is to be displayed. Possible values: • **cl_item_tree_model=>item_font_default**: Use the default font • **cl_item_tree_model=>item_font_fixed**: Use a fixed font • **cl_item_tree_model=>item_font_prop**: Use a proportional font
disabled **TYPE AS4FLAG**	Flag to indicate whether the item should be disabled (disabled items cannot be selected). Possible values: • **'X'**: Item is disabled • **' '**: Item can be selected
editable **TYPE AS4FLAG**	Flag to indicate whether the item can be edited. Possible values: • **'X'**: Item can be edited • **' '**: Item cannot be edited
hidden **TYPE AS4FLAG**	Flag to indicate whether the item is hidden. Possible values: • **'X'**: Item is hidden • **' '**: Item is not hidden

Structure TREEMLITEM

t_image **TYPE C(6)**	The image or icon to be displayed as part of the icon. Possible values: • `' '`: No icon • `'@XY@'`: The SAP icon with the code XY • **`'BNONE'`**: No icon. The text is shifted so that it begins in the position where the image would otherwise have been.
chosen **TYPE AS4FLAG**	Flag to indicate whether the item should be chosen. Possible values: • **`'X'`**: Chosen • `' '`: Not chosen
style **TYPE I**	Style of the item. Possible values: • **`cl_tree_model=>style_default`** • **`cl_tree_model=>style_emphasized`** • **`cl_tree_model=>style_emphasized_negative`** • **`cl_tree_model=>style_emphasized_positive`** • **`cl_tree_model=>style_inactive`** • **`cl_tree_model=>style_inherited`** • **`cl_tree_model=>style_intensified_critical`**
txtisqinfo **TYPE AS4FLAG**	Flag to indicate whether the quickinfo of the item should become its text. Possible values: • **`'X'`**: Quickinfo is used as the item text • `' '`: Quickinfo is not used as the item text
text **TYPE STRING**	Text of the item
alignment **TYPE I**	Alignment of the item. Possible values: • **`cl_item_tree_model=>align_left`** • **`cl_item_tree_model=>align_right`** • **`cl_item_tree_model=>align_auto`** The item is not aligned, but the display width is adjusted to the length of the item
length **TYPE I**	Length of the item in characters
ignoreimag **TYPE AS4FLAG**	Controls the width of the item. Possible values: • **`'X'`**: The length of the item is the length of the entire item. Icons occupy space that is then not available for text. • `' '`: The length of the item is the length of its text. Checkboxes and icons are then added to the length of the item.

usebgcolor **TYPE AS4FLAG**	• `'X'`: The background color of the item is slightly different from the background color of the control • `' '`: The background color of the item is the same color as the background color of the control.

Structure TREEMCITEM

Definition

TREEMCITEM is a structure that is used to define the line type of the internal table **TREEMCITAB**. This internal table is used in the Column Tree Model to specify the items that belong to a particular node.

Use

You can use **TREEMCITAB** to specify the type of the actual parameter you are going to pass to the **items_table** parameter in the **add_node** method of **cl_column_tree_model**.

Structure

Component and Type	Description
item_name **TYPE C(12)**	Name of the item
class **TYPE I**	Class of the item. Possible values: • **cl_item_tree_model=>item_class_text**: Item is a text • **cl_item_tree_model=>item_class_button**: Item is a pushbutton • **cl_item_tree_model=>item_class_checkbox**: Item is a checkbox • **cl_item_tree_model=>item_class_link**: Item is a link
font **TYPE I**	Font in which the item is to be displayed. Possible values: • **cl_item_tree_model=>item_font_default**: Use the default font • **cl_item_tree_model=>item_font_fixed**: Use a fixed font • **cl_item_tree_model=>item_font_prop**: Use a proportional font
disabled **TYPE AS4FLAG**	Flag to indicate whether the item should be disabled (disabled items cannot be selected). Possible values: • **'X'**: Item is disabled • **' '**: Item can be selected
editable **TYPE AS4FLAG**	Flag to indicate whether the item can be edited. Possible values: • **'X'**: Item can be edited • **' '**: Item cannot be edited
hidden **TYPE AS4FLAG**	Flag to indicate whether the item is hidden. Possible values: • **'X'**: Item is hidden • **' '**: Item is not hidden

t_image **TYPE C(6)**	The image or icon to be displayed as part of the icon. Possible values: • ' ': No icon • '@**XY**@': The SAP icon with the code XY • **'BNONE'**: No icon. The text is shifted so that it begins in the position where the image would otherwise have been.
chosen **TYPE AS4FLAG**	Flag to indicate whether the item should be chosen. Possible values: • **'X'**: Chosen • ' ': Not chosen
style **TYPE I**	Style of the item. Possible values: • **cl_tree_model=>style_default** • **cl_tree_model=>style_emphasized** • **cl_tree_model=>style_emphasized_negative** • **cl_tree_model=>style_emphasized_positive** • **cl_tree_model=>style_inactive** • **cl_tree_model=>style_inherited** • **cl_tree_model=>style_intensified_critical**
txtisqinfo **TYPE AS4FLAG**	Flag to indicate whether the quickinfo of the item should become its text. Possible values: • **'X'**: Quickinfo is used as the item text • ' ': Quickinfo is not used as the item text
text **TYPE STRING**	Text of the item

Structure TREEMCNODT

Definition

TREEMCNODT is a data structure that describes the attributes of a single node in a Column Tree Model. Its definition is stored centrally in the ABAP Dictionary, and you can use it to define the data types of your own parameters.

TREEMCNODT is also the line type of the internal table type **TREEMCNOTA**.

Use

You can use **TREEMCNODT** to specify the type of a work area for the actual parameter **node_table** in method **add_nodes** of class **cl_column_tree_model**.

Structure

Component	Type	Description
node_key	STRING	The key by which the node is identified in the tree. **This must be unique thorughout the tree**. You should only use letters, digits, and the underscore character in node keys.
relatkey	STRING	The key of a node to which the new node is related in position. If the new node is the first or last root node, this parameter must have the value `' '`.

relatship	`I`	The relationship between the new node and the node specified in `relatkey`. Possible values are: • `CL_TREE_MODEL=>RELAT_FIRST_CHILD` Inserts the new node as the first child node of the node specified in `relative_node_key`. This must be a folder. • `CL_TREE_MODEL=>RELAT_LAST_CHILD` Inserts the new node as the last child node of the node specified in `relative_node_key`. This must be a folder. • `CL_TREE_MODEL=>RELAT_PREV_SIBLING` Inserts the new node directly before the related node at the same level. • `CL_TREE_MODEL=>RELAT_NEXT_SIBLING` Inserts the new node directly after the related node at the same level. • `CL_TREE_MODEL=>RELAT_FIRST_SIBLING` Inserts the new node as the first node at the same level as the related node. • `CL_TREE_MODEL=>RELAT_LAST_SIBLING` Inserts the new node as the last node at the same level as the related node. Note: If `relatkey` is empty, the new node is inserted as a root node. Where the above values contain the word `FIRST` or `PREV`, it is inserted as the first root node. Where they contain `LAST` or `NEXT`, it is inserted as the last.
hidden	`AS4FLAG`	Specifies whether the node should be hidden (`'X'`) or visible (`' '`). Default is visible.
disabled	`AS4FLAG`	Specifies whether the node can be selected (`' '`) or not (`'X'`). The default is **not** disabled. **Note**: If a node is disabled, actions such as double-clicking it have no effect.
isfolder	`AS4FLAG`	Specifies whether the new node should be a folder or a leaf. Possible values: • `'X'`: Node is a folder `' '`: Node is a leaf

n_image	**C(6)**	Specifies the image used for the node. Possible values: • initial: The system uses the default values (leaf symbol for a leaf, closed folder symbol for a folder) • **'@XY@'**: An SAP icon with the code **XY**. • **'BNONE'**: No image is displayed. The node text begins at the position in which the image would normally be displayed. If you use this value for a node, you should also use it for all of its other same-level nodes.
exp_image	**C(6)**	Specifies the image used for an open folder. The possible values are the same as those listed above for the *image* parameter.
style	**I**	Sets the colors of the text and the background for the node. The possible values for this field are any static constant **CL_TREE_MODEL=>STYLE_***. For further details, refer to the definition of **CL_TREE_MODEL** in the Class Builder.
no_branch	**AS4FLAG**	Specifies whether connecting lines should be drawn between the nodes (**' '**) or not (**'X'**). The default is for the lines to be drawn.
expander	**AS4FLAG**	May only be set for a folder. If you set this attribute, the closed folder always displays a '+' symbol, even if it is empty. When the user clicks on the folder, the event **EXPAND_NO_CHILDREN** is triggered.
dragdropid	**I**	Only relevant if you want the node to be drag and drop-enabled. It contains the handle for a drag and drop object.
userobject	**REF TO OBJECT**	Can be assigned any reference to an application object
itemsincom	**AS4FLAG**	Indicates that the item specification is incomplete.

Structure TREEMSNOD

Definition

TREEMSNOD is a data structure that describes the attributes of a single node in a Column Tree Model. Its definition is stored centrally in the ABAP Dictionary, and you can use it to define the data types of your own parameters.

Structure

node_key	TM_NODEKEY	The key by which the node is identified in the tree. **This must be unique thorughout the tree**. You should only use letters, digits, and the underscore character in node keys.
relatkey	TM_NODEKEY	The key of a node to which the new node is related in position. If the new node is the first or last root node, this parameter must have the value ' '.
relatship	I	The relationship between the new node and the node specified in relatkey. Possible values are: • CL_TREE_MODEL=>RELAT_FIRST_CHILD Inserts the new node as the first child node of the node specified in relative_node_key. This must be a folder. • CL_TREE_MODEL=>RELAT_LAST_CHILD Inserts the new node as the last child node of the node specified in relative_node_key. This must be a folder. • CL_TREE_MODEL=>RELAT_PREV_SIBLING Inserts the new node directly before the related node at the same level. • CL_TREE_MODEL=>RELAT_NEXT_SIBLING Inserts the new node directly after the related node at the same level. • CL_TREE_MODEL=>RELAT_FIRST_SIBLING Inserts the new node as the first node at the same level as the related node. • CL_TREE_MODEL=>RELAT_LAST_SIBLING Inserts the new node as the last node at the same level as the related node. Note: If relatkey is empty, the new node is inserted as a root node. Where the above values contain the word FIRST or PREV, it is inserted as the first root node. Where they contain LAST or NEXT, it is inserted as the last.
hidden	AS4FLAG	Specifies whether the node should be hidden ('X') or visible (' '). Default is visible.

disabled	**`AS4FLAG`**	Specifies whether the node can be selected (`' '`) or not (`'X'`). The default is **not** disabled. **Note**: If a node is disabled, actions such as double-clicking it have no effect.
isfolder	**`AS4FLAG`**	Specifies whether the new node should be a folder or a leaf. Possible values: • `'X'`: Node is a folder `' '`: Node is a leaf
n_image	**`TV_IMAGE`**	Specifies the image used for the node. Possible values: • initial: The system uses the default values (leaf symbol for a leaf, closed folder symbol for a folder) • `'@XY@'`: An SAP icon with the code **`XY`**. • `'BNONE'`: No image is displayed. The node text begins at the position in which the image would normally be displayed. If you use this value for a node, you should also use it for all of its other same-level nodes.
exp_image	**`TV_IMAGE`**	Specifies the image used for an open folder. The possible values are the same as those listed above for the *image* parameter.
style	**`I`**	Sets the colors of the text and the background for the node. The possible values for this field are any static constant **`CL_TREE_MODEL=>STYLE_*`**. For further details, refer to the definition of **`CL_TREE_MODEL`** in the Class Builder.
no_branch	**`AS4FLAG`**	Specifies whether connecting lines should be drawn between the nodes (`' '`) or not (`'X'`). The default is for the lines to be drawn.
expander	**`AS4FLAG`**	May only be set for a folder. If you set this attribute, the closed folder always displays a '+' symbol, even if it is empty. When the user clicks on the folder, the event **`EXPAND_NO_CHILDREN`** is triggered.
dragdropid	**`I`**	Only relevant if you want the node to be drag and drop-enabled. It contains the handle for a drag and drop object.
userobject	**`REF TO OBJECT`**	Can be assigned any reference to an application object

Methods of the Control Framework

Methods of Class CL_GUI_CFW

The class `CL_GUI_CFW` contains static methods that apply to all instantiated custom controls when you call them.

dispatch

Use this method to dispatch application events (**see** Event Handling [Ext.]) to the event handlers registered for the events. If you do not call the method within the PAI event of your application program, it is called automatically by the system after the PAI has been processed. The method returns a return code from which you can tell if the call was successful.

```
CALL METHOD cl_gui_cfw=>dispatch
     IMPORTING return_code = return_code.
```

Parameters	Description
return_code	**`cl_gui_cfw=>rc_found`**: The event was successfully directed to a handler method. **`cl_gui_cfw=>rc_unknown`**: The event was not registered in the event list. **`cl_gui_cfw=>rc_noevent`**: No event was triggered in a control. The function code was therefore a normal one (for example, from a menu entry). **`cl_gui_cfw=>rc_nodispatch`**: No handler method could be assigned to the event.

⚠

An event can only be dispatched once. After that, it is "spent". Consequently, attempting to dispatch the events a second time does not trigger the handler events again.

flush

Use this method to synchronize the automation queue [Ext.]. The buffered operations are sent to the frontend using GUI RFC. At the frontend, the automation queue is processed in the sequence in which you filled it.

If an error occurs, an exception is triggered. You must catch and handle this error. Since it is not possible to identify the cause of the error from the exception itself, there are tools available in the Debugger and the SAPgui to enable you to do so.

Debugger: Select the option *Automation Controller: Always process requests synchronously*. The system then automatically calls the method **`cl_gui_cfw=>flush`** after each method called by the Automation Controller.

SAPGUI: In the SAPgui settings, under *Trace*, select *Automation*. The communication between the application server and the Automation Controller is then logged in a trace file that you can analyze at a later date.

```
CALL METHOD cl_gui_cfw=>flush
        EXCEPTIONS CNTL_SYSTEM_ERROR = 1
             CNTL_ERROR = 2.
```

⚠

Do not use any more synchronizations in your program than are really necessary. Each synchronization opens a new RFC connection to the SAPgui.

get_living_dynpro_controls

This method returns a list of reference variables to all active custom controls.

```
CALL METHOD cl_gui_cfw=>get_living_dynpro_controls
                  IMPORTING control_list = control_list.
```

Parameters	Description
`control_list`	List of reference variables of active custom controls. The list has the type `CNTO_CONTROL_LIST` (defined in class `CL_GUI_CFW`).

set_new_ok_code

You may only use this method in the handler method of a system event. It sets an **OK_CODE** that triggers PAI processing. This means that data is transferred from the screen to the program, and you can take control of the program in your PAI modules.

```
CALL METHOD cl_gui_cfw=>set_new_ok_code
        EXPORTING new_code = new_code
        IMPORTING     rc = rc.
```

Parameters	Description
new_code	Function code that you want to place in the **OK_CODE** field (**SY-UCOMM**).
return_code	**cl_gui_cfw=>rc_posted**: The OK_CODE was set successfully and the automatic field checks and PAI will be triggered after the event handler method has finished. **cl_gui_cfw=>rc_wrong_state**: The method was not called from the handler method of a system event. **cl_gui_cfw=>rc_invalid**: The **OK_CODE** that you set is invalid.

update_view

Calling the flush [Page 474] method only updates the automation queue if the queue contains return values.

If you have a queue with no return values, and want to ensure that it is synchronized, you can use the Control Framework method **CL_GUI_CFW=>UPDATE_VIEW**. You should only use this method if you absolutely need to update the GUI. For example, you might have a long-running application in which you want to provide the user with regular updates on the status of an action.

```
CALL METHOD cl_gui_cfw=>update_view
         EXCEPTIONS CNTL_SYSTEM_ERROR = 1
                CNTL_ERROR        = 2.
```

Methods of Class CL_GUI_OBJECT

The class **CL_GUI_OBJECT** contains important methods for custom control wrappers. The only one relevant for application programs is the is_valid [Page 479] method.

is_valid

This method informs you whether a custom control for an object reference still exists at the frontend.

```
CALL METHOD my_control->is_valid
        IMPORTING result = result.
```

Parameters	Description
result	0: Custom control is no longer active at the frontend 1: Custom control is still active

free

Use this method to destroy a custom control at the frontend. Once you have called this method, you should also initialize the object reference (**`FREE my_control`**).

```
CALL METHOD my_control->free
    EXCEPTIONS cntl_error        = 1
               cntl_system_error = 2.
```

Methods of Class CL_GUI_CONTROL

The class **CL_GUI_CONTROL** contains methods that you need to set control attributes (for example, displaying the control), register events, and destroy controls.

finalize

This method is redefined by the relevant control wrapper. It contains specific functions for destroying the corresponding control. This method is called automatically by the free [Page 480] method, before the control is destroyed at the frontend.

```
CALL METHOD my_control->finalize.
```

set_registered_events

Use this method to register the events of the control. **See also:** Event Handling [Ext.]

```
CALL METHOD my_control->set_registered_events
    EXPORTING  events          = events
    EXCEPTIONS cntl_error       = 1
           cntl_system_error = 2
      illegal_event_combination = 3.
```

Parameters	Description
events	Table of events that you want to register for the custom control **my_control**.

The table **events** is a list of the events that you want to register. It is defined with reference to table type **CNTL_SIMPLE_EVENTS**. The table type is based on the structure **CNTL_SIMPLE_EVENT**, which consists of the following fields:

Field	Description
EVENTID	Event name
APPL_EVENT	Indicates whether the event is a system event (initial) or an application event (X).

The values that you assign to the field **EVENTID** are control-specific and therefore described in the documentation of the individual controls.

get_registered_events

This method returns a list of all events registered for custom control `my_control`.

```
CALL METHOD my_control->get_registered_events
     IMPORTING  events      = events
     EXCEPTIONS cntl_error = 1.
```

Parameters	Description
events	Table of events that you want to register for the custom control `my_control`.

The table `events` is a list of the events that you want to register. It is defined with reference to table type `CNTL_SIMPLE_EVENTS`. The table type is based on the structure `CNTL_SIMPLE_EVENT`, which consists of the following fields:

Field	Description
EVENTID	Event name
APPL_EVENT	Indicates whether the event is a system event (initial) or an application event (X).

The values that you assign to the field `EVENTID` are control-specific and therefore described in the documentation of the individual controls.

For general information about event handling, refer to the Event Handling [Ext.] section of the SAP Control Framework documentation.

is_alive

This method informs you whether a custom control for an object reference still exists at the frontend.

```
CALL METHOD my_control->is_alive
    RETURNING state = state.
```

Parameters	Description
state	**`my_control->state_dead`**: Custom control is no longer active at the frontend **`my_control->state_alive`**: Custom control is active on the current screen. **`my_control->state_alive_on_other_dynpro`**: Custom control is not active on the current screen, but is still active (but invisible) at the frontend.

set_alignment

Use this method to align the custom control within its container:

```
CALL METHOD my_control->set_alignment
    EXPORTING  alignment        = alignment
    EXCEPTIONS cntl_error       = 1
           cntl_system_error = 2.
```

Parameters	Description
alignment	Control alignment

The **alignment** parameter may consist of combinations of the following alignments:

Name	Description
my_control->align_at_left	Alignment with left-hand edge
my_control->align_at_right	Alignment with right-hand edge
my_control->align_at_top	Alignment with top edge
my_control->align_at_bottom	Alignment with bottom edge

You can combine these parameters by adding the components:

alignment = my_control->align_at_left + my_control->align_at_top.

set_position

Use this method to place the control at a particular position on the screen.

⚠

The position of the control is usually determined by its container.

```
CALL METHOD my_control->set_position
    EXPORTING  height       = height
           left        = left
           top          = top
           width        = width
    EXCEPTIONS cntl_error       = 1
           cntl_system_error = 2.
```

Parameters	Description
height	Height of the control
left	Left-hand edge of the control
top	Top edge of the control
width	Width of the control

set_visible

Use this method to change the visibility of a custom control.

```
CALL METHOD my_control->set_visible
     EXPORTING  visible          = visible
     EXCEPTIONS cntl_error       = 1
                cntl_system_error = 2.
```

Parameters	Description
visible	`X`: Custom control is visible `' '`: Custom control is not visible

get_focus

This static method returns the object reference of the control that has the focus.

```
CALL METHOD cl_gui_control=>get_focus
    IMPORTING  control          = control
    EXCEPTIONS cntl_error       = 1
           cntl_system_error = 2.
```

Parameters	Description
control	Object reference (**`TYPE REF TO cl_gui_control`**) to the control that has the focus.

set_focus

Use this static method to set the focus to a custom control.

```
CALL METHOD cl_gui_control=>set_focus
    EXPORTING  control          = control
    EXCEPTIONS cntl_error        = 1
           cntl_system_error = 2.
```

Parameters	Description
control	Object reference (**`TYPE REF TO cl_gui_control`**) to the control on which you want to set the focus.

get_height

This method returns the height of the control.

```
CALL METHOD control->get_height
    IMPORTING  height          = height
    EXCEPTIONS cntl_error       = 1.
```

Parameters	Description
height	Current height of the control

get_width

This method returns the width of the control.

```
CALL METHOD control->get_width
    IMPORTING  width            = width
    EXCEPTIONS cntl_error       = 1.
```

Parameters	Description
width	Current width of the control

Methods of the Class CL_DRAGDROP

The class **CL_DRAGDROP** contains methods that describe the drag and drop [Page 106] behavior of a custom control.

constructor

The constructor creates an instance for the description of the drag and drop behavior of a control.

```
CREATE OBJECT dragdrop.
```

add

This method adds a new description to the drag and drop behavior. You can store any number of descriptions, but you may not add the same description more than once.

```
CALL METHOD dragdrop->add
   EXPORTING flavor        = flavor
         dragsrc       = dragsrc
         droptarget    = droptarget
         effect        = effect
         effect_in_ctrl = effect_in_ctrl
   EXCEPTIONS already_defined  = 1
          obj_invalid      = 2.
```

Parameters	Description
flavor	Description of the new flavor
dragsrc	'`X`': The description is a drag source
droptarget	'`X`': The description is a drop target
effect	Drop effect of the description between different custom controls. The following effects are supported: `dragdrop->copy`: Appearance of the mouse when using drag and drop to copy. `dragdrop->move`: Appearance of the mouse when using drag and drop to move. `dragdrop->none`: Drag and drop is not possible.
effect_in_ctrl	Drop effect of the description in the same custom control. The following effects are supported: `dragdrop->copy`: Appearance of the mouse when using drag and drop to copy. `dragdrop->move`: Appearance of the mouse when using drag and drop to move. `dragdrop->none`: Drag and drop is not possible. `dragdrop->use_default_effect`: Uses the same effect specified in the `effect` parameter.

Exceptions	Description
already_defined	The specified flavor has already been defined.
obj_invalid	The object has already been destroyed using the method destroy [Page 498].

➡

If you use the **`copy`** and **`move`** effects when you define the flavor, the system uses the **`move`** effect when the user drags an object normally, and the **`copy`** effect when the user presses and holds the CTRL key while dragging.

clear

Deletes the contents of the instance. Once you have called this method, you cannot perform any more drag and drop operations on the corresponding custom control.

```
CALL METHOD dragdrop->clear
    EXCEPTIONS obj_invalid = 1.
```

Exceptions	Description
obj_invalid	The object has already been destroyed using the method destroy [Page 498].

destroy

Deletes the contents of the instance. The instance itself is also destroyed. Once you have called this method, you cannot perform any more drag and drop operations on the corresponding custom control.

CALL METHOD dragdrop->destroy.

get

Returns the complete description of a flavor.

```
CALL METHOD dragdrop->get
    EXPORTING  flavor        = flavor
    IMPORTING  isdragsrc      = isdragsrc
               isdroptarget  = isdroptarget
               effect        = effect
               effect_in_ctrl = effect_in_ctrl
    EXCEPTIONS not_found    = 1
               obj_invalid = 2.
```

Parameters	Description
flavor	Name of the flavor
dragsrc	'**X**': The description is a drag source
droptarget	'**X**': The description is a drop target
effect	Drop effect of the description between different custom controls. The following effects are supported: **dragdrop->copy**: Appearance of the mouse when using drag and drop to copy. **dragdrop->move**: Appearance of the mouse when using drag and drop to move. **dragdrop->none**: Drag and drop is not possible.
effect_in_ctrl	Drop effect of the description in the same custom control. The following effects are supported: **dragdrop->copy**: Appearance of the mouse when using drag and drop to copy. **dragdrop->move**: Appearance of the mouse when using drag and drop to move. **dragdrop->none**: Drag and drop is not possible. **dragdrop->use_default_effect**: Uses the same effect specified in the **effect** parameter.

Exceptions	Description
already_defined	The specified flavor has already been defined.

➡

If you use the **copy** and **move** effects when you define the flavor, the system uses the **move** effect when the user drags an object normally, and the **copy** effect when the user presses and holds the CTRL key while dragging.

get_handle

This method returns the handle of the drag and drop position. In most cases, you will not need to use this method. However, for tabular mass data interfaces (such as the SAP Tree), you must copy this handle into the interface table.

```
CALL METHOD dragdrop->get_handle
   IMPORTING  handle = handle
   EXCEPTIONS obj_invalid = 1.
```

Parameters	Description
handle	Handle of the drag and drop description

Exceptions	Description
obj_invalid	The object has already been destroyed using the method destroy [Page 498].

modify

Use this method to change an existing flavor.

```
CALL METHOD dragdrop->modify
   EXPORTING  flavor        = flavor
              dragsrc       = dragsrc
              droptarget    = droptarget
              effect        = effect
              effect_in_ctrl = effect_in_ctrl
   EXCEPTIONS not_found    = 1
              obj_invalid = 2.
```

Parameters	Description
flavor	Name of the flavor
dragsrc	'**X**': The description is a drag source
droptarget	'**X**': The description is a drop target
effect	Drop effect of the description between different custom controls. The following effects are supported: **dragdrop->copy**: Appearance of the mouse when using drag and drop to copy. **dragdrop->move**: Appearance of the mouse when using drag and drop to move. **dragdrop->none**: Drag and drop is not possible.
effect_in_ctrl	Drop effect of the description in the same custom control. The following effects are supported: **dragdrop->copy**: Appearance of the mouse when using drag and drop to copy. **dragdrop->move**: Appearance of the mouse when using drag and drop to move. **dragdrop->none**: Drag and drop is not possible. **dragdrop->use_default_effect**: Uses the same effect specified in the **effect** parameter.

Exceptions	Description
not_found	The specified flavor does not exist
obj_invalid	The object has already been destroyed using the method destroy [Page 498].

If you use the **copy** and **move** effects when you define the flavor, the system uses the **move** effect when the user drags an object normally, and the **copy** effect when the user presses and holds the CTRL key while dragging.

remove

Use this method to delete a flavor.

```
CALL METHOD dragdrop->remove
    EXPORTING  flavor = flavor
    EXCEPTIONS not_found   = 1
            obj_invalid = 2.
```

Parameters	Description
flavor	Name of the flavor

Exceptions	Description
not_found	The specified flavor does not exist
obj_invalid	The object has already been destroyed using the method destroy [Page 498].

Methods of the Class CL_DRAGDROPOBJECT

The class CL_DRAGDROPOBJECT describes the context of a drag and drop operation [Page 106]. It contains information about the source object, the flavor of the drag and drop operation, and information about the source and target.

set_flavor

You can only use this method within event handling for the ONGETFLAVOR event. Use the **`newflavor`** parameter to determine the flavor that you want to use in the drag and drop operation. You receive a list of available flavors as an event parameter.

```
CALL METHOD dragdropobject->set_flavor
    EXPORTING newflavor = newflavor
    EXCEPTIONS illegal_state  = 1
            illegal_flavor = 2.
```

Parameters	Description
newflavor	Name of the flavor

Exceptions	Description
invalid_state	You did not call the method from within event handling for **`ONGETFLAVOR`**.
obj_invalid	You used a flavor that is not supported by the current drag and drop situation.

abort

Terminates the drag and drop operation immediately. No further events are triggered.

CALL METHOD dragdropobject->abort.