Tree Select Plugin for Oracle APEX

Technical Documentation Developed by MOHAMMAD ALQURAN

Version 1.0 – November 2025

1. Overview

The **Tree Select Plugin for Oracle APEX** is a reusable region plugin that renders a hierarchical tree with checkboxes, tri-state support, search, lazy loading, and a context menu. It is designed to work with Universal Theme and can use either a static JSON data source or SQL / table-based sources via the plugin's PL/SQL AJAX interface.

The plugin is intended mainly for scenarios where users must select multiple items from a hierarchical structure, such as category trees, organizational charts, or role / privilege hierarchies.

2. Architecture

The plugin consists of three main layers:

- **PL/SQL package (xx_tree_select_pkg)** — renders the base region markup, exposes plugin attributes, and responds to AJAX requests (load roots, load children, search, batch children).
- **JavaScript module (xx_tree_select.js)** — handles initialization, node rendering, selection and tri-state logic, bulk operations, search, and the context menu / info dialog.
- **CSS stylesheet (xx_tree_select.css)** — defines layout, colors, typography, connector lines, and optional dark theme styling.

Initialization is performed by a global function `xxTreeSelectInit(cfg)`, which is called from the PL/SQL render procedure using `apex_javascript.add_onload_code`.

3. Initialization & Configuration

The `xxTreeSelectInit(cfg)` function receives a configuration object built in PL/SQL. Important properties include:

- `staticId` — region Static ID used to locate the DOM node.
- `ajaxIdentifier` — plugin AJAX identifier passed to `apex.server.plugin`.
- `storeItem / storeFormat` — page item and format (`CSV` or `JSON`) to store the selection.
- `allowParent`, `allowLeaf`, `triState` — rules for which nodes are selectable.
- `defaultExpand` — number of levels to expand on initial render.
- `searchMode` — `NONE`, `CLIENT`, or `SERVER`.
- `hasIcons`, `compact`, `theme` — visual options.

During initialization the script:

1. Resolves the region DOM element by `staticId`.
2. Builds an in-memory cache for nodes (by id and by parent).
3. Loads and renders root nodes using either static JSON or AJAX.
4. Wires toolbar buttons (`.ts-expandAll`, `.ts-collapseAll`, `.ts-checkAll`, `.ts-uncheckAll`).
5. Attaches the search box and hotkeys.
6. Configures the context menu and info dialog.
7. Initializes the hidden store item to reflect the current selection.

4. Data Normalization & Caching

Different server-side data sources (SQL, table, static JSON) may expose slightly different column names. The JavaScript layer normalizes raw rows into a consistent internal model with fields such as:

- `ID` , `PARENT_ID` — the hierarchical relationship (root nodes have `PARENT_ID` = NULL).
- `LABEL` — the text displayed for each node.
- `IS_DISABLED` — `'Y'` or `'N'` , controls whether a node is selectable.
- `ICON` — optional Font Awesome icon class name for the node.

At runtime the JavaScript module maps these columns into its internal model (for example `id` , `parentId` , `label` , `disabled` , `icon` and derived flags such as `isLeaf` or `selected`), but from the developer perspective all data sources share the same external column contract.

The cache used per region typically holds:

- a map of `id → row` ,
- a map of `parentId → child id list` ,
- flags that indicate which nodes have been loaded from the server.

> This cache enables client-side operations like expand / collapse and client-mode search without additional round trips.

5. Rendering & Layout

Rendering is handled by helper functions that generate HTML for nodes and lists. These functions build `<li>` elements with classes like `ts-node` , plus data attributes such as `data-node` (node id) and `data-level` (depth).

Each node label typically contains:

- a toggle button, when the node has children,
- a checkbox (`.ts-check`) when selection is allowed,
- optional icon (`.ts-icon`) and accent strip,
- a text span (`.ts-text`) where matched text can be highlighted.

Layout of toolbar + tree is defined in CSS (e.g. `.ts-layout` , `.ts-rail` , `.ts-main`). The current design supports a vertical button rail on the left and the search + tree on the right.

6. Selection & Tri-State Logic

Selection state is reflected through a combination of checkbox properties and CSS classes (`.is-checked` , `.is-mixed`). The main rules:

- Leaf nodes follow the user's direct input.
- Children propagate their state upward via a helper (commonly named `recalcUpwards`).
- Parents can be:
  - **checked** — all checkable children are checked;
  - **unchecked** — no checkable child is checked or mixed;
  - **mixed** — any other combination.
- Disabled nodes are not toggled by bulk operations and are ignored when computing parent state.

A separate helper (commonly `recomputeAllFromLeaves`) performs a bottom-up recompute of all parents after bulk operations, by sorting nodes in order of depth and applying `recalcUpwards` from leaves to root.

After each change, the list of selected node ids is written into the store item: a CSV string or JSON array, depending on configuration.

7. Bulk Operations

Toolbar buttons provide bulk operations:

- **Expand all / Collapse all** operate on the current tree structure.
- **Check all / Uncheck all** operate on enabled checkboxes only.

Two behaviors exist for bulk selection:

- **Normal click** — applies to the nodes currently rendered / visible.
- **SHIFT + click** — can optionally preload all descendants from the server (via a `children_batch` call or per-node requests) before applying the bulk operation.

After toggling the desired nodes, the script invokes a single bottom-up recompute, updates the store item, and may trigger a custom event on the region so that Dynamic Actions can react.

8. Search (Client & Server Modes)

The search box (`.ts-search`) is wired by a helper function (often named `attachSearch`) and supports two modes:

Client Mode

- All nodes are available in the client cache.
- The script filters the tree entirely in JavaScript.
- A helper converts SQL-like patterns (using `%` and `_`) into JavaScript regular expressions so wildcard search behaves naturally.
- Matched segments in labels are wrapped in `<mark class="ts-hit">`.

Server Mode

- The search term is sent to the PL/SQL `ajax` procedure.
- The server applies a `WHERE ... LIKE` filter to the data source.
- The resulting nodes (plus ancestors) are rendered client-side.
- The same highlighting logic is used so results visually match the server filter.

Clearing the search term restores either the full root set or the previous view, depending on configuration.

9. PL/SQL Plugin Package

The PL/SQL package (e.g. `xx_tree_select_pkg`) implements the plugin specification and body. Key responsibilities:

- **render** — emits the region markup (toolbar, search, empty `<ul>`) and injects the JavaScript initialization call with attribute values.
- **ajax** — inspects the requested action and returns JSON via `APEX_JSON` . Typical actions:
    - `get_roots`
    - `get_children`
    - `search`
    - `children_batch` (optional optimization)

- **Attribute mapping** — reads plugin attributes such as data source type, store format and visual options using helper APIs (e.g. `apex_plugin_util.get_varchar2_attribute` in newer APEX versions).

Sample tables (like `demo_tree_table`) illustrate how to provide hierarchical data through `id`, `parent_id`, and `label` columns.

10. Dynamic Actions & Events

The region can be integrated with Dynamic Actions in multiple ways:

- DA on the underlying store item (when its value changes, refresh reports or charts).
- DA on a custom event triggered by the tree (e.g., when selection changes or when a node is clicked).
- Standard region events like After Refresh can be used to re-wire custom logic if necessary.

This enables building reactive pages, such as faceted search interfaces or dashboards that respond to selections in the tree.

11. Styling & Theming

All markup is wrapped in a region container with classes like `t-TreeSelect` and `ts-dark`. Within this, the script emits structural classes such as `.ts-layout`, `.ts-rail`, `.ts-main`, `.ts-node`, `.ts-label`, `.ts-icon`, and `.ts-toggle`.

The default CSS implements a modern look with dark panel backgrounds, connector lines, hover states, and optional colored accents. Since class names are stable, applications can override or extend the CSS at the theme or page level without modifying plugin resources.

> For large applications, consider centralizing overrides in a dedicated theme style or app-level CSS file, keeping plugin upgrades simpler.

12. Compatibility & Migration

The plugin targets Oracle APEX 19.2 and newer, using Universal Theme. It relies on:

- `apex.server.plugin` for AJAX calls,
- APEX plugin attribute APIs for reading configuration,
- jQuery and core APEX JavaScript namespaces.

Newer APEX releases (e.g. 23.x, 24.x) add improved helper APIs and stricter CSP settings. For long-term maintenance:

- Consider updating attribute access code to use `apex_plugin_util.get_varchar2_attribute` and related helpers.
- Verify that static file references for JS and CSS are still correct after upgrades.
- Test CSS alignment with the current Universal Theme version.

If compatibility mode changes, re-test AJAX actions and ensure the JSON shapes consumed by the JavaScript layer remain stable.

Features and Capabilities Overview

Tree Functionality

- Full tri-state checkbox tree integrated with Oracle APEX page items and Dynamic Actions.
- Parent / child recompute logic that respects disabled nodes and supports mixed states.
- Dynamic expand and collapse per node, plus toolbar buttons to expand/collapse all.

- Client and Server search modes, both supporting SQL-style wildcards (`%` and `_`).
- Shift-click style bulk operations (Check All / Uncheck All) for fast selection changes.
- Optimized behavior for large data sets with optional lazy loading of child nodes.

User Interface & Styling

- Toolbar with expand, collapse, check, uncheck, and search controls.
- Flexible layout that supports faceted-style pages (search above, controls on the side of the tree).
- Per-node icons using the `ICON` column, fully themeable via CSS.
- Universal Theme–friendly colors and typography so the tree blends with the rest of the application.
- Accessible markup using `role="tree"` , `aria-expanded` , and `aria-hidden` where appropriate.
- Self-contained dialog behavior that avoids interfering with the page's scroll or global overlays.

Integration & Configuration

- Implemented as a declarative APEX region plugin, configured entirely from the builder.
- Supports the plugin's custom Dynamic Actions such as "Tree Select - Changed", "Tree Select - Search", and "Tree Select - Loaded".
- Works with multiple data source types (SQL Query, SQL Table, Static JSON, and REST-based sources via APEX).
- Designed for modern APEX 21+ with Universal Theme, and can be configured to work with older APEX versions when needed.
- No external JavaScript libraries required; the plugin ships with its own JS and CSS bundle.

In short, the Tree Select Plugin provides a rich, maintainable, and APEX-native way to present and manage hierarchical data with checkboxes, while remaining flexible enough to adapt to many different use cases (filtering reports, faceted search, navigation trees, and more).

13. Future Enhancements

These features are planned for future versions of the Tree Select Plugin:

- **Expand / Collapse State Memory** — remember which branches were expanded and restore them automatically on page refresh.
- **Improved Accessibility & Keyboard Navigation** — refine focus order and ARIA usage for screen readers.
- **Optional Drag-and-Drop Reordering** — allow reordering of nodes with persistence through AJAX.

Features such as lazy loading of child nodes and per-node icons from the `ICON` column are already supported in the current version.

Tree Select Plugin for Oracle APEX
Developed by **Mohammad Alquran**
Contact: moh.alquraan@gmail.com

Donate

Chat on WhatsApp