Textbox

Overview

This is a single line text entry field. The initialization value is set by the 'value' property. The 'value' property is also used to receive input in a Rich UI application through field binding.

Validation (Rich UI Only)

Please visit the Validation and Error Messages page.

Field Binding Dialog (Rich UI Only)

Please visit the Field Binding page.

Input Type (Rich UI Only)

The 'input type' property can be used to set the 'type' attribute of the HTML <input> element that is rendered for the textbox. Input types are primarily useful for mobile applications, as the mobile device will display a different version of the keypad based on the value (i.e. numeric entry, telephone number, etc.) However, some of these options are supported by HTML5 capable browsers. If this field is not set a standard textbox element will be used by default.

Date - The Date input type is used to create a date field for the textbox. When supported it allows your user to select a date which is automatically formatted for the field and returns the date as a value.

Datetime - Similar to the Date input type this allows the user to select a date and time (with time zone).

Email - This input type would be used for a field you are expecting an email address to be entered. For mobile browsers this field allows your device to recognize the email type and change the on-screen keyboard to match (giving @ and .com options depending on the device).

Time - The time input type gives the user a field to accept a time entry format. This input type does not give a time zone like the datetime type.

Month - The month input type functions like the Date type, with the exception there is no day indicated in the format. It allows the user to only select a month and year.

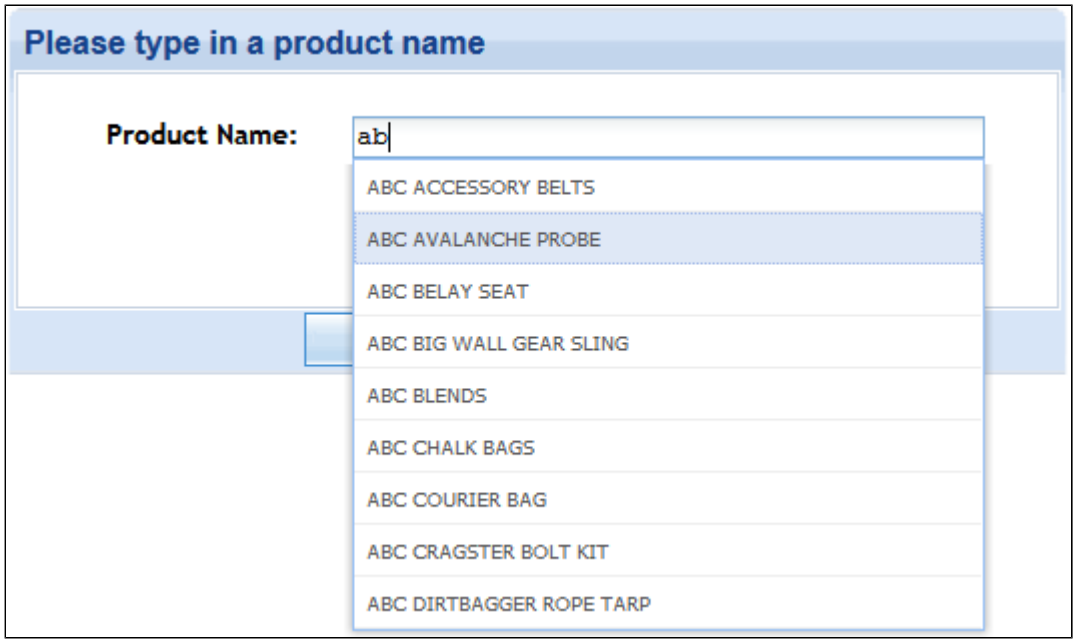
Number - The number type should be used when a field is set to contain a numeric value you want to impose a restriction on.

Tel - This input type would be used for a field needing a telephone number format.

URL - The url type is for input fields containing a URL address. For mobile browsers this field allows your device to recognize the email type and change the on-screen keyboard to match (giving .com options depending on the device).

Auto-Complete Choices

Auto-complete allows your textbox field to automatically suggest choices based on text entered into the textbox.



The auto-complete choices/values can be populated multiple ways:

1. 'choices/values' properties

- ***Choice option field*** - Is where your fields are selected in order how you would like the auto-complete to display them.
- ***Choice values field*** - Is where the value you want returned to your application is set.

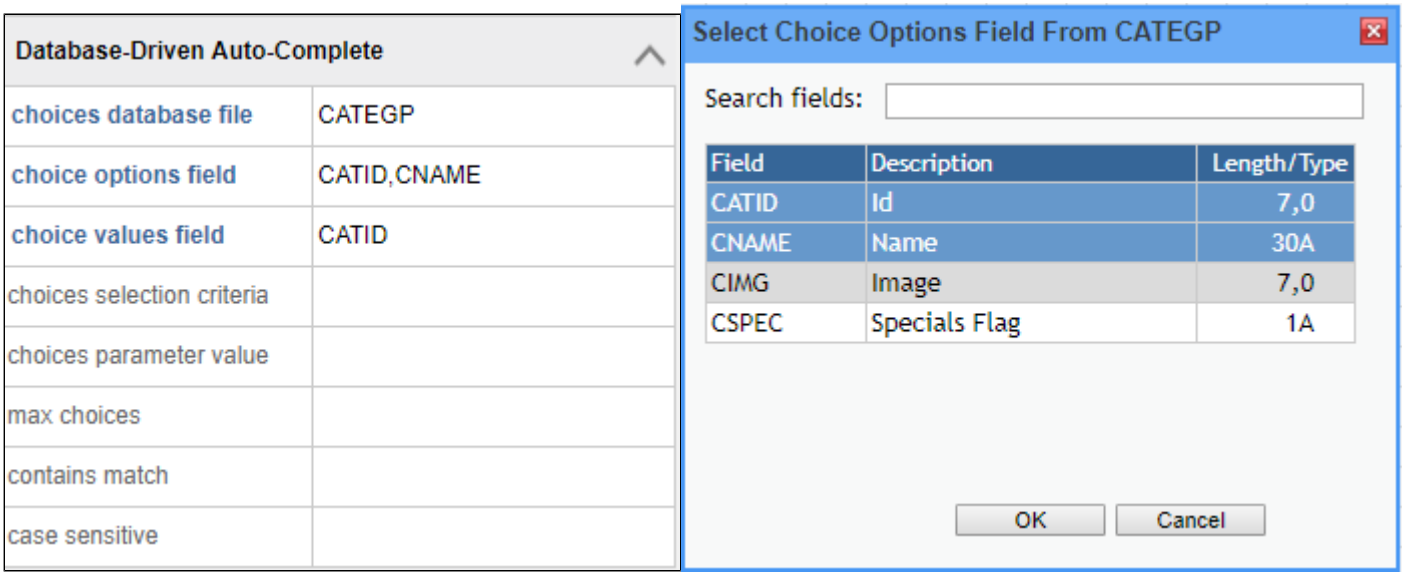
Here are two ways to populate the choice option/values properties:

1. ***Comma separated list*** - A simple list of values separated by just a comma (i.e. Apple, Orange, Banana).
2. ***JSON array format*** - Using JSON array format, meaning the choices/values are enclosed in a bracket with quotations and then comma separated (i.e. ["ElementOne", "ElementTwo"]) within the choices and choice values fields also allows you to populate the auto-complete, these fields can also be bound to an RPG field (**Rich UI only**).

2. Database-Driven Auto-Complete - See Database-Driven Auto-Complete section below.

Database-Driven Auto-Complete

This section allows choices/values to be retrieved from a database file.

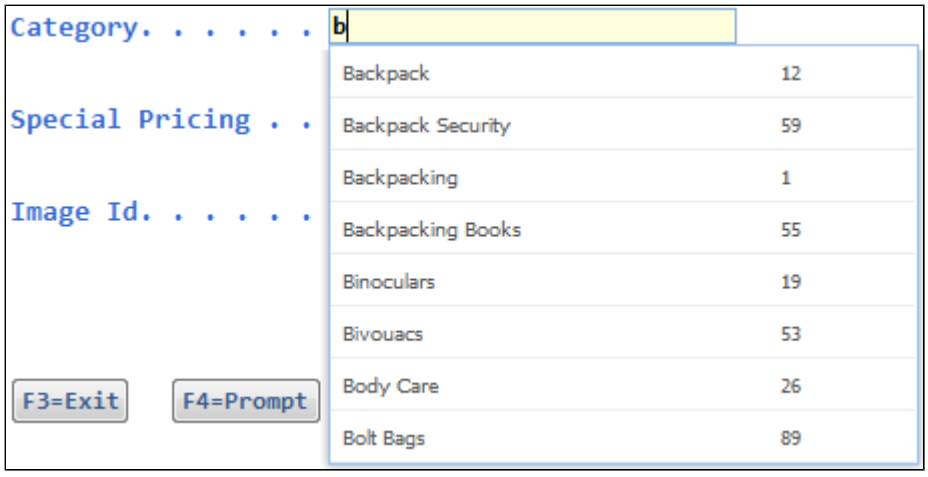


Choices database file - Enter the database file to use to populate your options/values. When setting your database file it's not required to qualify with library name, although you can. if the library is omitted, the application job's library list will be used.

Choice options field - Select the field(s) you wish to use to search. If there are multiple fields specified in the choice options field this will allow you to return multiple values in the auto-complete search. However, only the first value in the field will be used to search, any other selected fields are for display only as additional columns.

Choice values field - Select the field to return a value to your application.

In the example, records will be loaded from file CATEGP. The records will be searched on field CNAME and CATID, with field CATID displaying in a second column (see below). The value in field CATID in the selected record will be returned to the program.



SQL Expressions

You can also use SQL expressions to return information in addition to just selecting database fields from the file. So in the example, using the expression **TRIM(DBLNAM) || ', ' || DBFNAM** allows us to return a concatenated search value with formatting for our search, but the returned value to the application is unchanged.

This is what the SQL expression returns:

Max Choices
The max choices property allows you to set the number of choices you will see from your database-driven selection. There is no limit to the number of choices you can show, however if there is no limit set the default is 10.

Contains Match

This property when set to true looks for records that contain your search text. When set to false the query looks for your at the start of the records to match your search text. The default for this is set to false.

Dynamic Auto-Complete

Choices URL

The choices URL property allows you to use an external program to return your choice options and values by passing them using JSON formatting. A PHP script would be one example of a custom program used to pass the values to your application. When using the choices URL property, all database-driven auto-complete properties are ignored.

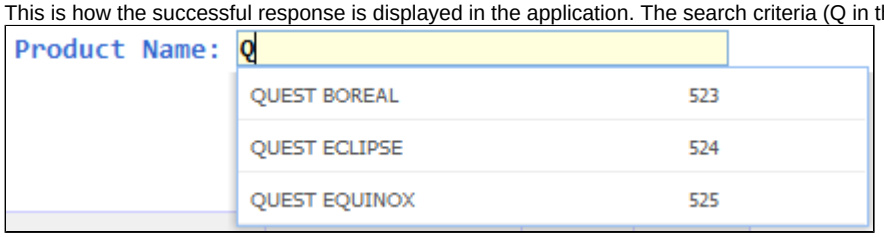
JSON Successful Response Example

These are values returned from an auto-complete textbox in a successful response from the external program:

```
{
    "success": true,
    "response": {
        "results": [
            {
                "PNAME": "QUEST BOREAL",
                "PRID": "523"
            },
            {
                "PNAME": "QUEST ECLIPSE",
                "PRID": "524"
            },
            {
                "PNAME": "QUEST EQUINOX",
                "PRID": "525"
            }
        ],
        "colWidths": [
            30,
            10
        ]
    }
}
```

In the successful response you can see the fields **PNAME**, **PRID** are being set by the script and the values returned in the successful JSON. The additional **colWidths** is setting the column size of the widget when multiple fields are being returned (this is optional).

This is how the successful response is displayed in the application. The search criteria (Q in the example below) is posted to the program as field **query**.



JSON Error Response Example

This example is JSON information returned when the script encounters an error with an external program driven auto-complete.

```
{
    "success": false,
    "errorId": "08001",
    "errorText": "Authorization failure on distributed database connection attempt. SQLCODE=-30082"
}
```

If you are unable to get any information to display on your choice URL script you can use the showErrors() API to try and debug the reason the auto-complete is unable to display results.
showErrors() API Documentation

Example of the showErrors() API displaying the JSON error response example:

Full PHP Script Example:

```php
<?php

  define("MAX", 10);
  define("DB", "S100450A");
  define("USER", "user");
  define("PASS", "password");

  $records = array();
  $query = "";
  $limit = 0;
  if (isset($_REQUEST["query"])) $query = $_REQUEST["query"];
  if (isset($_REQUEST["limit"])) $limit = intval($_REQUEST["limit"]);

  if ($query != "") {

    $query = strtoupper($query) . "%";
    if ($limit == 0 || $limit > MAX) $limit = MAX;

    // Example of error reporting.
    $con = db2_connect(DB, USER, PASS, array("i5_naming" => DB2_I5_NAMING_ON));
    if (!$con) {

      $response = array(

        "success" => false,
        "errorId" => db2_conn_error(),
        "errorText" => db2_conn_errormsg()

      );

      print json_encode($response);
      return;

    }

    // The following will not be error checked, for brevity.
    $stm = "select distinct pname, prid from rpgspcart/prodp where pname like ? order by pname";
    $stm = db2_prepare($con, $stm);
    db2_bind_param($stm, 1, "query", DB2_PARAM_IN, DB2_CHAR);
    db2_execute($stm);

    $count = 0;
    while (db2_fetch_row($stm) && $count < $limit) {

      $record = array();
      $record["PNAME"] = trim(db2_result($stm, "PNAME"));
      $record["PRID"] = trim(db2_result($stm, "PRID"));

      $records[$count++] = $record;

    }

  }

  // "colWidths" is optional, this helps the widget to
  // size the columns when multiple fields are displayed.

  $return = array(

    "success" => true,
    "response" => array (
      "results" => $records,
      "colWidths" => array(
        db2_field_precision($stm, "PNAME"),
        db2_field_precision($stm, "PRID")
      )
    )

  );

  print json_encode($return);

?>
```

Auto-Complete Results Template

The auto-complete results panel is built using a preset HTML template that is built into the product. CSS classes are attached to allow for custom styling. For complete control over the output, a custom HTML template can be assigned using the "results template" property. This property accepts an HTML fragment that will be inserted into the results panel for each returned record. In the HTML template, a field name within parenthesis will be replaced with the actual field data when the results HTML is generated.

For example, to make each record display as a block with 2 lines that display one field each, the "results template" property could be set to the following:

```
<div class="autocomplete-item">
  <div class="autocomplete-col"><strong>Label one:</strong> (MYFIELD)</div>
  <div class="autocomplete-col"><strong>Label two:</strong> (MYFIELD2)</div>
</div>
```

The CSS classes assigned can then be used to control styling such as borders, etc.