

TStringList: The Swiss Army Knife You're Probably Underusing

By Dr. Holger Flick

If you've been writing Delphi code for any length of time, you've used `TStringList`. It's the go-to container for managing lists of strings. But here's the thing: most developers only scratch the surface of what this versatile class can do.

Let me show you three powerful features that often fly under the radar: duplicate handling, built-in sorting, and the surprisingly useful `CommaText` property.

Duplicate Handling: Keep Your Lists Clean

Did you know `TStringList` can automatically prevent duplicates? The `Duplicates` property gives you fine-grained control over how duplicate entries are handled.

```
var
  List: TStringList;
begin
  List := TStringList.Create;
  try
    List.Duplicates := dupIgnore; // Silently ignore duplicates
    List.Add('Apple');
    List.Add('Banana');
    List.Add('Apple'); // This won't be added

    ShowMessage(List.Text); // Shows only "Apple" and "Banana"
  finally
    List.Free;
  end;
end;
```

The `Duplicates` property has three options:

- `dupIgnore`: Silently rejects duplicate entries (my personal favorite for building unique lists)
- `dupAccept`: Allows duplicates freely (the default behavior)
- `dupError`: Raises an `EStringListError` exception when you try to add a duplicate

Here's a practical example - collecting unique email domains from a user list:

```

procedure GetUniqueDomains(Users: TStringList; Domains: TStringList);
var
    Email, Domain: string;
    AtPos: Integer;
begin
    Domains.Clear;
    Domains.Duplicates := dupIgnore;
    Domains.Sorted := True; // Must be sorted for dupIgnore to work!

    for Email in Users do
    begin
        AtPos := Email.IndexOf('@');
        if AtPos > 0 then
        begin
            Domain := Email.Substring(AtPos + 1).ToLower;
            Domains.Add(Domain);
        end;
    end;
end;

// Usage:
var
    Users, Domains: TStringList;
begin
    Users := TStringList.Create;
    Domains := TStringList.Create;
    try
        Users.Add('john@company.com');
        Users.Add('jane@company.com');
        Users.Add('bob@example.org');
        Users.Add('alice@company.com');

        GetUniqueDomains(Users, Domains);
        ShowMessage(Domains.Text); // company.com, example.org
    finally
        Users.Free;
        Domains.Free;
    end;
end;

```

Important gotcha: Duplicate detection only works when `Sorted` is `True`. The duplicate check relies on binary search, which requires a sorted list.

Built-in Sorting: No Need to Reinvent the Wheel

Speaking of sorting, `TStringList` has built-in sorting capabilities that are criminally underused.

```

var
    Names: TStringList;
begin
    Names := TStringList.Create;
    try
        Names.Add('Zimmerman');
        Names.Add('Anderson');
        Names.Add('Baker');

        Names.Sort; // Manual sort
        ShowMessage(Names.Text); // Anderson, Baker, Zimmerman
    finally
        Names.Free;
    end;
end;

```

But here's where it gets interesting - you can enable automatic sorting:

```
Names.Sorted := True; // Automatically maintains sort order
Names.Add('Davis');   // Inserted in correct position automatically
```

When `Sorted` is `True`, every `Add` operation automatically places the string in the correct sorted position. This is perfect for maintaining alphabetically ordered lists without manual intervention.

Custom Sorting with `OnCompare`

Need case-insensitive sorting? Or maybe you want to sort by string length? Use the `CustomSort` method:

```
var
  List: TStringList;
begin
  List := TStringList.Create;
  try
    List.Add('zebra');
    List.Add('Apple');
    List.Add('banana');

    // Case-insensitive sort
    List.CustomSort(
      function(List: TStringList; Index1, Index2: Integer): Integer
      begin
        Result := CompareText(List[Index1], List[Index2]);
      end
    );

    ShowMessage(List.Text); // Apple, banana, zebra
  finally
    List.Free;
  end;
end;
```

Or sort by string length:

```
List.CustomSort(
  function(List: TStringList; Index1, Index2: Integer): Integer
  begin
    Result := Length(List[Index1]) - Length(List[Index2]);
  end
);
```

The `CustomSort` comparison function works exactly like other comparison functions in Delphi - it should return a negative value if the first item should come before the second, zero if they're equal, and a positive value if the first item should come after the second. For example, `CompareText(List[Index1], List[Index2])` returns -1 when Index1's string is alphabetically before Index2's string, 0 when they're equal, and 1 when Index1 comes after Index2. You can use simple arithmetic for numeric comparisons like `Length(List[Index1]) - Length(List[Index2])` which naturally produces the correct negative/zero/positive result for sorting by string length.

CommaText: The Serialization Shortcut

The `CommaText` property is a hidden gem for quickly serializing and deserializing string lists. It automatically handles comma-separated values with proper quoting for strings that contain spaces or special characters.

```

var
  Config: TStringList;
begin
  Config := TStringList.Create;
  try
    Config.Add('Server');
    Config.Add('Port');
    Config.Add('User Name'); // Contains a space

    // Serialize to comma-separated text
    ShowMessage(Config.CommaText);
    // Output: Server,Port,"User Name"

    // Deserialize from comma-separated text
    Config.Clear;
    Config.CommaText := 'Database,Timeout,"Connection String"';
    ShowMessage(Config.Text);
    // Shows three separate lines
  finally
    Config.Free;
  end;
end;

```

Notice how "User Name" and "Connection String" are automatically quoted because they contain spaces? That's `CommaText` handling the complexity for you.

Real-World Use Case: Configuration Files

Here's a practical example - saving and loading application settings:

```

procedure SaveRecentFiles(const RecentFiles: TStringList);
var
  IniFile: TIniFile;
begin
  IniFile := TIniFile.Create(ChangeFileExt(ParamStr(0), '.ini'));
  try
    IniFile.WriteString('Settings', 'RecentFiles', RecentFiles.CommaText);
  finally
    IniFile.Free;
  end;
end;

procedure LoadRecentFiles(RecentFiles: TStringList);
var
  IniFile: TIniFile;
begin
  IniFile := TIniFile.Create(ChangeFileExt(ParamStr(0), '.ini'));
  try
    RecentFiles.CommaText := IniFile.ReadString('Settings', 'RecentFiles', '');
  finally
    IniFile.Free;
  end;
end;

```

Understanding Delimiters

You're not stuck with commas. `TStringList` has `Delimiter` and `DelimitedText` properties for custom separators:

```

var
    List: TStringList;
begin
    List := TStringList.Create;
    try
        List.Delimiter := '|';
        List.StrictDelimiter := True; // Don't treat spaces as delimiters
        List.DelimitedText := 'Alpha|Beta|Gamma Delta';

        ShowMessage(List[2]); // "Gamma Delta" - space preserved
    finally
        List.Free;
    end;
end;

```

The `StrictDelimiter` property is crucial here - without it, spaces are treated as additional delimiters, which can cause unexpected splitting.

Putting It All Together

Here's a real-world example combining all three features - building a unique, sorted tag list from comma-separated input:

```

function BuildTagList(const Input: string): string;
var
    Tags: TStringList;
begin
    Tags := TStringList.Create;
    try
        Tags.Sorted := True;           // Enable auto-sorting
        Tags.Duplicates := dupIgnore;  // Reject duplicates
        Tags.CommaText := Input;       // Parse input

        Result := Tags.CommaText;      // Return cleaned, sorted, unique list
    finally
        Tags.Free;
    end;
end;

// Usage:
var
    CleanTags: string;
begin
    CleanTags := BuildTagList('delphi, pascal, delphi, programming, Pascal');
    ShowMessage(CleanTags); // "delphi,pascal,programming"
end;

```

The Bottom Line

`TStringList` is far more capable than most developers realize. Before you reach for a third-party collection library or write custom sorting code, check if `TStringList` already does what you need:

- Use `Duplicates` to maintain unique lists automatically
- Use `Sorted` for automatic alphabetical ordering
- Use `CommaText` for quick serialization and parsing
- Use `CustomSort` when you need specialized sorting logic

These features have been in Delphi since the early days, rock-solid and battle-tested. Sometimes the best tools are the ones hiding in plain sight.

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

[Support on Ko-fi —](#)