NG Drag&Drop Guide

USER MANUAL

LMD Innovative

This page is intentionally left blank.
Remove this text from the manual
template if you want it completely blank.

This page is intentionally left blank.
Remove this text from the manual
template if you want it completely blank.

Overview

1 Overview

LMD NG Drag&Drop is a part of Next Generation (NG) package suite. All these packages are based on new IDE and language features of latest Delphi IDE versions.

NG Drag&Drop provides the ability to exchange data with other applications via standard Windows OLE drag&drop protocol. There are a lot of common applications, which uses Windows drag&drop:

- Windows Explorer (shell) - allows to drag/receive real or virtual file objects.
- MS Office application (Word, Excel, Outlook, ect), WordPad, other text editors - allows to drag/receive ANSI, Unicode, RTF and HTML text data, images, metafiles, files.
- Internet Browsers (Internet Explorer, Google Chrome, ect) - allows to drag/receive URLs (to open new page or as a bookmark), text (as search query to open new page), files (as downloads).
- Adobe Photoshop - allows to drag/receive images.
- many other high quality Windows applications actually support drag&drop.

OLE drag&drop protocol declares two sides, which participate in data exchange:

- Source side provides the data and initiate data dragging. Please look at Data Dragging as Source section.
- Target side receives dragging data, and handle data drop. Please look at Data Accepting as Target section.

NG Drag&Drop supports both sides, and so, the application, which uses the package, can act as data source or (and) as data target. Dragging data is called data object, and its a collection of user data stored in predefined formats. There are a lot of standard data formats, which allows to send/receive text, RTF, HTML, images, files, URLs, and are built-in NG Drag&Drop package. The following standard/common formats are implemented:

- `CF_TEXT` and `CF_UNICODETEXT` - for dragging ANSI and Unicode text data.
- `CF_BITMAP` and `CF_DIB` - for dragging bitmaps.
- `CF_ENHMETAFILE` and `CF_METAFILEPICT` - for dragging Windows metafiles.
- `RTF` - for RTF text.
- `HTML` - for HTML text.
- `INETURLA` and `INETURLW` - for ANSI and Unicode URLs.
- `CF_HDROP` - for dragging real existing files.
- `FILEDESCRIPTOR` and `FILECONTENTS` - for dragging virtual files.

Users application, acting as a data source, can configure data object to include any desired combination of data formats. As well, application acting as a target can support any desired formats combination, accepting only required formats.

NG Drag&Drop provides a way for declaring custom formats. Custom formats have they own unique names, and mostly used inside the application to transfer specifically formatted application data or to prevent other applications to receive dragging data. Custom formats can be implemented on top of any other built-in formats, just overriding format's name; or, from the scratch, by overriding data reading and writing methods.

Please look at Data Formats section to learn more about supported data formats.

Components

NG Drag&Drop include the following components, accessible from the Delphi's component palette at design-time:

- `TNGDropSource` - allows to configure dragging data and initiate drag&drop operation (source side).
- `TNGDropTarget` - allows to receive dragging data by registering some application's control as a drop target and providing related events like `OnDragEnter`, `OnDragOver`, `OnDragLeave`, `OnDrop`; and a single `OnDragAction` event, which can be used instead of all previously mentioned events to simplify code.

Also, each supported data format is represented as a class descendant from `TNGDataFormat` base class, for example, `TNGTextFormat`, `TNGUnicodeTextFormat`, `TNGBitmapFormat`, ect. Moreover, each built-in data format class has its shortcut alias like `CF.TEXT`, `CF.UNICODETEXT`, `CF.BITMAP`, which are declared to make user's code more understandable.

Fluent Interface

NG Drag&Drop provides Fluent Interface API , which is an easy and very convenient way for supporting drag&drop operations, without placing any component on the form. API supports both source and target parts. For example, data dragging (source part) can be initiated as follows:

```
NGDropSource.AddText('My text')
            .AddUnicodeText('My text')
            .Execute;
```

Features

Following is a short feature list of NG Drag&Drop package:

- `TNGDropSource` component, which allows to drag data from customer's application to any other drag&drop enabled applications.
- `TNGDropTarget` component, which allows to receive data from any drag&drop enabled application.
- Formal dealing of data formats. Any data drag operation can include any number of formats in dragged data object. There no restrictions of format combinations used. Any drop target as well can be configured to accept any combination of data formats.
- Built-in standard and common formats implementation:
  - `CF_TEXT`
  - `CF_UNICODETEXT`
  - `CF_BITMAP`
  - `CF_DIB`
  - `CF_ENHMETAFILE`
  - `CF_METAFILEPICT`
  - `RTF`
  - `HTML`
  - `INETURLA` (Ansi)
  - `INETURLW` (Unicode)
  - `CF_HDROP`
  - `FILEDESCRIPTOR`
  - `FILECONTENTS`
- `TNGDataFormat` base data format class can be subclassed for advanced implementation of complex custom formats. Simple custom formats could be defined declaratively on the top of any existing format class using `CustomFormat` attribute.
- Data format types shortcuts, such as `CF.TEXT` for `TNGTextFormat`, or `CF.RTF` for `TNGRtfFormat`.

This page is intentionally left blank.
Remove this text from the manual
template if you want it completely blank.

Data Dragging as Source

2 Data Dragging as Source

Configuring Dragging Data

`TNGDropSource` component allows to initiate dragging operation. Dragging data should be configured first using `TNGDropSource.Data` property. The property provide access to `TNGSourceData` object, which declares many methods to manipulate dragging data: `Add`, `AddText`, `AddUnicodeText`, `AddBitmap`, `AddDib`, `Clear`, `Remove`, `HasFormat`, `HasAny`, `Count` and `Items`.

The most important methods are `Add` and `AddXXX` methods, which allows to add some data to drop source. Formally, any data can be added via `Add` method using required data format class:

```
NGDropSource1.Data.Add(TNGTextFormat.Data('My dragging text'));
```

Since built-in formats has shortcuts in the form of CF.XXX, the code can be clarified:

```
NGDropSource1.Data.Add(CF.TEXT.Data('My dragging text'));
```

Moreover, special `AddXXX` methods are provided for some formats to make code even more simpler:

```
NGDropSource1.Data.AddText('My dragging text');
```

To provide high level of control, NG Drag&Drop allows to work with data formats formally, independently and explicitly. So, any combination of required data formats can be added to dragging data:

```
NGDropSource1.Data.Add(CF.URL.Data('http://www.google.com'));
NGDropSource1.Data.AddText('http://www.google.com');
NGDropSource1.Data.AddUnicodeText('http://www.google.com');
NGDropSource1.Data.Add(CF.FILEDESCRIPTOR.Data(...));
NGDropSource1.Data.Add(CF.FILECONTENTS.Data(...));
```

The above example shows dragging data configuration to allow dragging URL to browser (via CF.URL format), text editors (via CF.TEXT and CF.UNICODETEXT formats) and to Windows file explorer, creating a file link to web page (via `CF.FILEDESCRIPTOR and CF.FILECONTENTS`) formats. Please look at Data Formats section for more information about data supported formats.

Performing Drag&Drop Operation

After source data has been configured the drag&drop operation can be executed. To begin drag&drop operation `Execute` method of `TNGDropSource` component should be called:

```
NGDropSource1.Execute;
```

The method acts like modal dialogs executing methods and do not return until the drag&drop operation ends. Typically, drag&drop operation should be started from some control's `OnMouseDown` event handle, that is, when the mouse button is down. In usual cases its not a good idea to initiate drag&drop operation from `OnClick` event handler, because at this point mouse button is already up.

```
procedure TForm1.Panel1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  NGDropSource1.Data.AddText('My dragging text');
  NGDropSource1.Execute;
end;
```

NG Drag&Drop provides automatic detecting of mouse buttons state change; it remembers the state at the beginning of the drag and cancel drag&drop operation, if state is changed. This default algorithm can be customized using `OnQueryContinueDrag` event.
Also, drag&drop operation is canceled when Escape key is pressed.

Usually, dragging data is configured for each drag&drop operation independently, so, `TNGDropSource` component provides `AutoClear` property, which is set to `True` by default, and forces the component to clear dragging data after each `Execute` method call.

`Execute` method allows to specify allowed drop effects, which can include `deMove`, `deCopy` or `deLink`. If the parameter is omitted, all of these effects are allowed. The method returns the actual effect, which has been chosen by the target during drag&drop operation, or - `deNone`, if the operation has been canceled. To learn more about drop effects please read Drop Effects section.

Fluent Interface

Since drag&drop operation executions are usually tiny and contains only few lines of code, NG Drag&Drop provides special API for executing drag&drop operations even without placing the component on the form. The API is provided by `NGDropSource` global function, which returns special `TNGDropSource.TBuilder` object and can be used like this:

```
NGDropSource.AddText('My dragging text')
            .AddUnicodeText('My dragging text')
            .Execute;
```

It contains, actually, the same `Add`, `AddXXX` and `Execute` methods as in previously discussed `TNGDropSource` component.

This page is intentionally left blank.
Remove this text from the manual
template if you want it completely blank.

Data Accepting as Target

3 Data Accepting as Target

Accepting data

`TNGDropTarget` component allows to accept dragging data. the component provides `Control` property, which should be set to link to some `TWinControl` placed on the form. After that, this control become registered as a drag&drop target, and can accept data when the data dragged over this control. `TNGDropTarget` component provides a lot of events for controlling current drag&drop operation:

- `OnDragEnter` event is fired when the mouse cursor enter the control area.
- `OnDragOver` event is fires when the mouse cursor moves over the control area.
- `OnDragLeave` event is fired when the mouse cursor leaves the control area.
- `OnDrop` event is fired when the user drops dragging data (e.g. when he release mouse button).

All these events declares "`C`" parameter of type `TNGTargetContext`, which contains all information about current drag&drop operation state, and provides methods and properties to accept or reject dragging data. The following members can be used to query current drag&drop operation state:

- `C.Action` property determines current drag action: `daEnter`, `daOver`, `daLeave` or `daDrop`.
- `C.KeyState` property allows to determine, which mouse buttons are currently down and whether Shift, Ctrl or Alt keys are pressed.
- `C.CursorPos` property allows to determine current mouse cursor position in screen coordinates.
- `C.Allowed` property provides access to allowed drop effects, which has been specified by the operation source side. In `daDrop` action this property specifies currently accepted drop effect; please read below. To learn more about drop effects please read Drop Effects section.
- `C.Data` property provides access to `TNGTargetData` object, which can be used to query currently dragging data. The data contains a lot of properties and methods, like: `HasFormat`, `HasAny`, `Formats`, `FormatName`, `AsFormat`, `AsText`, `AsUnicodeText`, `AsBitmap`, ect.

Based on information, provided by the properties described above, the code in event handlers should decide, whether to accept data or not. If dragging data should be accepted by current drop target, `C.Accepted` context property should be set to preferred drop effect. the following rules apply to `C.Accepted` and `C.Allowed` properties in different drag&drop events:

- `daEnter` (`OnDragEnter`) - `C.Allowed` is set to allowed by the source drop effects. `C.Accepted` is set to `deNone` initially, and can be changed inside event handler; however, its value is not really used, because, for simplicity, `OnDragOver` event is fired immediately after `OnDragEnter`.
- `daOver` (`OnDragOver`) - `C.Allowed` is set to allowed by the source drop effects. `C.Accepted` is set to `deNone` initially, and can be changed inside event handler; this value is used to setup mouse cursor to indicate current accept state.
- `daLeave` (`OnDragLeave`) - `C.Allowed` is set to allowed by the source drop effects. `C.Accepted` is set to `deNone`, and **cannot** be changed inside event handler. `C.Data` is also not accessible in this event.
- `daDrop` (`OnDrop`) - `C.Allowed` is set to **previously chosen** in the `OnDragOver` event drop effect. `C.Accepted` is also set to this drop effect initially and **cannot** be changed.

These rules implies the following: `C.Accepted` should be really set only in `daOver` (`OnDragOver`) event. `daDrop` (`OnDrop`) - will not be fired is the data is not accepted, e.g. `C.Accepted = deNone`.

Since `C.Accepted` property cannot be set to value not included in `C.Allowed` property, its tricky to specify value for it manually. So, `TNGTargetContext` class provides a set of overloaded `Accept` methods, which takes different parameters and can be used to simplify code. In simplest form `Accept` method can take no parameters, which means that it accept data without any condition choosing from allowed drop

effects automatically, based on currently pressed keys (Shift, Ctrl, Alt). To learn more about drop effects please read Drop Effects section.

`TNGDropTarget` component also provides `OnDragAction` event, which can be used instead of previously described events, and allows to simlify source code by having whole drag&drop related code inside a single event.

Lets now show an example of data accepting using `OnDragAction` event. The simplest case will look like:

```
procedure TForm1.NGDropTarget1DragAction(Sender: TObject; C: TNGTargetContext);
begin
  case C.Action of
    daOver: if C.Data.HasFormat(CF.TEXT) then
              C.Accept;
    daDrop: Edit1.Text := C.Data.AsText;
  end;
end;
```

Accept Helper Methods

`TNGTargetContext` object provides a set of overloaded `Accept` helper methods. The simplest case without any parameters was discussed above. All of these methods was specifically designed to allow very easy implementation of drop target event handlers. To get the impression of how they simplify code, lets write another example:

```
procedure TForm1.NGDropTarget1DragAction(Sender: TObject; C: TNGTargetContext);
var
  s: AnsiString;
begin
  if C.AcceptText(s) then
    Edit1.Text := s;
end;
```

That's all. No even "`case C.Action of`" is really required. The following rules are applied to `C.Accept` helper methods to allow such a simple code writing:

- All these methods has `Accepted` parameter, which specifies possible accepted drop effects, but can be omitted, since it has a default value, indicating that all possible drop effects can be accepted. To learn more about drop effects please read Drop Effects section.
- If the data should be accepted, all these methods sets `C.Accepted` to appropriate value, which is determined automatically, based on `C.KeyState`, `C.Allowed` and method's `Accepted` parameter value.
- In actions other than `daDrop`, **all these methods returns False**, independently of whether the data can be accepted or not; and return `True` only if the data has been really accepted (in `daDrop` event only). They has been specially implemented in this way to support the following usage scenario:

```
procedure TForm1.NGDropTarget1DragAction(Sender: TObject; C: TNGTargetContext);
var
  s:  AnsiString;
  us: string;
begin
  if C.AcceptUnicodeText(us) then
    Edit1.Text := s
  else if C.AcceptText(s) then
    Edit1.Text := s
end;
```

That is:
- in `daOver` event both formats (CF.TEXT and CF.UNICODETEXT) have a chance to be accepted.
- In `daDrop` event, Unicode text (if available in dragging data) will be preferred to ANSI text.

Please note, that `C.Accept` helper methods has been designed to simplify code in simple cases. In more advances cases, all other methods, described below can be used explicitly.

Fluent Interface

Dragging data acceptance can be tiny as well as drag&drop operations execution. For such cases NG Drag&Drop provides special API for configuring drop targets even without placing the component on the form. The API is provided by `NGDropTarget` global function, which returns special `TNGDropTarget.TBuilder` object and can be used like this:

```
procedure TForm6.FormCreate(Sender: TObject);
begin
  NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
  var
    s: AnsiString;
  begin
    if C.AcceptText(s) then
      Edit1.Text := s;
  end);
end;

procedure TForm6.FormDestroy(Sender: TObject);
begin
  NGDropTarget.Unregister(MyTargetPanel);
end;
```

`NGDropTarget.Register` method can be used to register some `TWinControl` as a drop tasrget, and its usually called from `OnFormCreate` event handler. Do not forget to unregister drop target calling `NGDropTarget.Unregister` method eventually.

Drop Effects

4 Drop Effects

When the data is dragged over some drag&drop target, the target should accept or reject the data. But, in OLE drag&drop its not just a Boolean value, the target should specify which drop effect it prefers. The following drop effects are predefined by OLE drag&drop interface:

- Move (`deMove`) - data is moved from one place to another, or from one application to another. After successful drag&drop operation initial data should be removed.
- Copy (`deCopy`) - data is copied from one place to another, or from one application to another. After successful drag&drop operation initial data should not be removed, and two copies of data should exist as a result.
- Link (`deLink`) - some sort of link to initial data should be created as a result of successful drag&drop operation.
- None (`deNone`) - target do not accept dragging data.

Most people are familiar with drop effects, because they are used in Windows Explorer when dragging files. If the file is simply dragged it will be moved from one location to another. If the user holds Ctrl key pressed, mouse cursor include "+" glyph to indicate that dragging file will be copied. As well, if the user holds Ctrl+Shift keys pressed, a link to dragging file will be created.

The same thing actually happens when the user drag selected text from WordPad to Word, for example. If the Ctrl key is not pressed the text will be "moved" from one application to another, disappearing in the initial application. Of course, what is really happens, is that receiving application adds dragging text to its document, and then sending (source) application remove dragged text from its document.

Some applications can restrict possible drop effects. For example, bookmarks in Google Chrome can not be copied, they can be only moved.

When initiating drag&drop operation, the source side should specify, which drop effects are allowed for this operation. `TNGDropSource.Execute` method allows to specify allowed drop effects using its `Allowed` parameter. If the value for this parameter is not provided, all drop effects will be allowed.
If the target side want to accepts dragging data it should choose one drop effect from allowed effects to indicate data acceptance. Context's `C.Accepted` property should be set to chosen drop effect or to `deNone`, if the target do not accept dragging data.
After dragging data has been accepted and dropped on the target, `TNGDropSource.Execute` returns and provide the final drop effect, chosen by the target, as a result value.

Choosing drop effect can be tricky task for target, since it need to analyze provided by the source allowed drop effects, analyze `C.KeyState`, analyze its own possibilities (e.g. whether it can support, for example, `deLink` at all). So, NG Drag&Drop provides helper `Accept` methods, which can simplify the task. `Accept` methods provide `Accepted` parameter, which can be used to specify, which drop effects are supported by the target. If the parameter value is not specified, all drop effects will be accepted.

NG Drag&Drop use the following algorithm to choose drop effect:

- Intersect allowed by the source drop effects with supported by the target.
- If intersected effects include `deLink`, and Ctrl+Shift keys are pressed, then the resulting drop effect is `deLink`.
- Otherwise, if intersected effects include `deCopy`, and Ctrl key is pressed, then the resulting drop effect is `deCopy`.
- Otherwise, if intersected effects include `deMove`, then the resulting drop effect is `deMove`.
- Otherwise, if intersected effects include `deCopy`, then the resulting drop effect is `deCopy`, even if no keys are pressed.

- Otherwise, if intersected effects include `deLink`, then the resulting drop effect is `deLink`, even if no keys are pressed.
- Otherwise the resulting drop effect is `deNone`.

So, in very simple case `C.Accept` method can be called to accept dragging data without worrying about effects, and NG Drag&Drop will choose appropriate drop effect automatically. In more advanced cases, when the application wants to customize the above algorithm, it should set the value of `C.Accepted` property manually, without using `Accept` helper methods.

This page is intentionally left blank.
Remove this text from the manual
template if you want it completely blank.

Data Formats

5 Data Formats

To transfer data between applications the data should be formatted in memory in some standard way to allow different applications understand each other. NG Drag&Drop provides support for most standard and common data formats, used in OLE drag&drop to drag and receive data to/from many Windows applications. This includes text, RTF, HTML, pictures, URLs, files data and more.

Typically single drag&drop operation transfer data object, which contain data in several data formats simultaneously. For example, `CF_TEXT` and `CF_UNICODETEXT` data are commonly transferred together to support both: Unicode and non-Unicode applications. As well, some formats are specially designed to be used together, like `FILEDESCRIPTOR` and `FILECONTENTS` formats, which are used to drag/receive virtual files. Unlike most other libraries, NG Drag&Drop allow to work with formats formally, by providing the possibility to include any combination of data formats into single drag&drop operation.

In NG Drag&Drop each data format is represented by a class descendant from `TNGDataFormat` base class. Following is a list of all built-in data formats:

- TNGTextFormat
- TNGUnicodeTextFormat
- TNGBitmapFormat
- TNGDibFormat
- TNGEnhMetafileFormat
- TNGMetafilePictFormat
- TNGRtfFormat
- TNGHtmlFormat
- TNGUrlFormat
- TNGHDropFormat
- TNGFileDescriptorFormat and TNGFileContentsFormat

As well, NG Drag&Drop provides the ability to declare custom data formats. This feature can be used to implement application's private drag&drop formats or to implement missing commonly used formats. To learn about custom data formats please read the Custom Formats section.

NG Drag&Drop demo includes a feature to enumerate data formats in currently dragged data object. This feature can be used to learn, which applications supports which data formats.

5.1 TNGTextFormat

`TNGTextFormat` class implements standard `CF_TEXT` data format and allows to drag/receive ANSI text data. NG Drag&Drop use `AnsiString` standard Delphi type for working with ANSI text data.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag text data at the source side, while `Ref` method can be used to receive text data at the target side. NG Drag&Drop declares special type alias `CF.TEXT` to make user's code more readable:

```
NGDropSource.Add(CF.TEXT.Data('My dragging text'))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: AnsiString;
begin
  if C.Accept(CF.TEXT.Ref(@s)) then
    Edit1.Text := string(s);
end);
```

Moreover, special methods are declared for working with text data, which can simplify code even more:

```
NGDropSource.AddText('My dragging text')
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: AnsiString;
begin
  if C.AcceptText(@s) then
    Edit1.Text := string(s);
end);
```

5.2 TNGUnicodeTextFormat

`TNGUnicodeTextFormat` class implements standard `CF_UNICODETEXT` data format and allows to drag/receive Unicode text data. NG Drag&Drop use `string` standard Delphi type for working with Unicode text data.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag text data at the source side, while `Ref` method can be used to receive text data at the target side. NG Drag&Drop declares special type alias `CF.UNICODETEXT` to make user's code more readable:

```
NGDropSource.Add(CF.UNICODETEXT.Data('My dragging text'))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: string;
begin
  if C.Accept(CF.UNICODETEXT.Ref(@s)) then
    Edit1.Text := s;
end);
```

Moreover, special methods are declared for working with text data, which can simplify code even more:

```
NGDropSource.AddUnicodeText('My dragging text')
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: string;
begin
  if C.AcceptUnicodeText(@s) then
    Edit1.Text := s;
end);
```

5.3 TNGBitmapFormat

`TNGBitmapFormat` class implements standard `CF_BITMAP` data format and allows to drag/receive bitmaps. NG Drag&Drop use `TBitmap` standard Delphi type for working with bitmap data.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag image data at the source side, while `Ref` method can be used to receive images at the target side. NG Drag&Drop declares special type alias `CF.BITMAP` to make user's code more readable:

```
NGDropSource.Add(CF.BITMAP.Data(b))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  b: TBitmap;
begin
  b := Tbitmap.Create;
  try
    if C.Accept(CF.BITMAP.Ref(b)) then
      ShowBitmap(b);
  finally
    b.Free;
  end;
end);
```

Moreover, special methods are declared for working with bitmap data, which can simplify code even more:

```
NGDropSource.AddBitmap(b)
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  b: TBitmap;
begin
  b := Tbitmap.Create;
  try
    if C.AcceptBitmap(b) then
      ShowBitmap(b);
  finally
    b.Free;
  end;
end);
```

5.4 TNGDibFormat

`TNGDibFormat` class implements standard `CF_DIB` data format and allows to drag/receive device independed bitmaps. NG Drag&Drop use `TBitmap` standard Delphi type for working with bitmap data.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag image data at the source side, while `Ref` method can be used to receive images at the target side. NG Drag&Drop declares special type alias `CF.DIB` to make user's code more readable:

```
NGDropSource.Add(CF.DIB.Data(b))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  b: TBitmap;
begin
  b := Tbitmap.Create;
  try
    if C.Accept(CF.DIB.Ref(b)) then
      ShowBitmap(b);
  finally
    b.Free;
  end;
end);
```

Moreover, special methods are declared for working with bitmap data, which can simplify code even more:

```
NGDropSource.AddDib(b)
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  b: TBitmap;
begin
  b := Tbitmap.Create;
  try
    if C.AcceptDib(b) then
      ShowBitmap(b);
  finally
    b.Free;
  end;
end);
```

5.5 TNGEnhMetafileFormat

`TNGEnhMetafileFormat` class implements standard `CF_ENHMETAFILE` data format and allows to drag/receive Windows GDI metafiles. NG Drag&Drop use `TMetafile` standard Delphi type for working with metafile data.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag metafile data at the source side, while `Ref` method can be used to receive metafiles at the target side. NG Drag&Drop declares special type alias `CF.ENHMETAFILE` to make user's code more readable:

```
NGDropSource.Add(CF.ENHMETAFILE.Data(m))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  m: TMetafile;
begin
  m := TMetafile.Create;
  try
    if C.Accept(CF.ENHMETAFILE.Ref(m)) then
      ShowMetafile(m);
  finally
    m.Free;
  end;
end);
```

Moreover, special methods are declared for working with bitmap data, which can simplify code even more:

```
NGDropSource.AddEnhMetafile(m)
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  m: TMetafile;
begin
  m := TMetafile.Create;
  try
    if C.AcceptEnhMetafile(m) then
      ShowMetafile(m);
  finally
    m.Free;
  end;
end);
```

5.6 TNGMetafilePictFormat

`TNGMetafilePictFormat` class implements standard `CF_METAFILEPICT` data format and allows to drag/receive old style Windows metafiles. NG Drag&Drop use `TMetafile` standard Delphi type for working with metafile data.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag metafile data at the source side, while `Ref` method can be used to receive metafiles at the target side. NG Drag&Drop declares special type alias `CF.METAFILEPICT` to make user's code more readable:

```
NGDropSource.Add(CF.METAFILEPICT.Data(m))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  m: TMetafile;
begin
  m := TMetafile.Create;
  try
    if C.Accept(CF.METAFILEPICT.Ref(m)) then
      ShowMetafile(m);
  finally
    m.Free;
  end;
end);
```

5.7 TNGRtfFormat

`TNGRtfFormat` class implements common RTF data format and allows to drag/receive Rich Text Format data. As specified, RTF data is sequence of 7-bit ASCII chars, and so, NG Drag&Drop use `RawByteString` standard Delphi type for working with RTF.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag RTF data at the source side, while `Ref` method can be used to receive RTF at the target side. NG Drag&Drop declares special type alias `CF.RTF` to make user's code more readable:

```
NGDropSource.Add(CF.RTF.Data(s))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: RawByteString;
begin
  if C.Accept(CF.RTF.Ref(@s)) then
    ShowRtf(s);
end);
```

Please look at provided demo project to understand how to get RTF data from `TRichEdit` Delphi component, and how to set it back.

5.8 TNGHtmlFormat

`TNGHtmlFormat` class implements common HTML data format and allows to drag/receive HTML data. NG Drag&Drop use `string` standard Delphi type for working with HTML.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag HTML data at the source side, while `Ref` method can be used to receive HTML at the target side. NG Drag&Drop declares special type alias `CF.HTML` to make user's code more readable:

```
NGDropSource.Add(CF.HTML.Data(s))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: string;
begin
  if C.Accept(CF.HTML.Ref(@s)) then
    ShowHtml(s);
end);
```

HTML format is bit tricky, and contain special text headers, so please look MSDN Documentation to understand it. NG Drag&Drop does not add/parse such header data internally, so it should be added manually.
Please, look at provided demo project to understand how to work with HTML data format.

5.9 TNGUrlFormat

`TNGUrlFormat` class implements common URL data format and allows to drag/receive ANSI URLs. NG Drag&Drop use `AnsiString` standard Delphi type for working with ANSI URLs.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag URL data at the source side, while `Ref` method can be used to receive text data at the target side. NG Drag&Drop declares special type alias `CF.URL` to make user's code more readable:

```
NGDropSource.Add(CF.URL.Data('http://google.com'))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: AnsiString;
begin
  if C.Accept(CF.URL.Ref(@s)) then
    Edit1.Text := string(s);
end);
```

5.10 TNGUrlWFormat

`TNGUrlWFormat` class implements common URL data format and allows to drag/receive Unicode URLs. NG Drag&Drop use `string` standard Delphi type for working with Unicode URLs.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag URL data at the source side, while `Ref` method can be used to receive text data at the target side. NG Drag&Drop declares special type alias `CF.URLW` to make user's code more readable:

```
NGDropSource.Add(CF.URLW.Data('http://google.com'))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: string;
begin
  if C.Accept(CF.URLW.Ref(@s)) then
    Edit1.Text := s;
end);
```

5.11 TNGHDropFormat

`TNGHDropFormat` class implements standard CF_HDROP data format and allows to drag/receive really existing files. NG Drag&Drop use `TNGStrArray` type, which is a dynamic array of string, for working with CF_HDROP data. The data should contain one or more really existing file paths.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag file paths data at the source side, while `Ref` method can be used to receive text data at the target side. NG Drag&Drop declares special type alias `CF.HDROP` to make user's code more readable:

```
SetLegnth(s, 3);
s[0] := 'c:\MyText1.txt';
s[0] ;= 'c:\MyText2.txt';
s[0] := 'c:\MyText3.txt';

NGDropSource.Add(CF.HDROP.Data(s))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: TNGStrArray;
begin
  if C.Accept(CF.HDROP.Ref(@s)) then
    ShowFiles(s);
end);
```

5.12 TNGFileDescriptorFormat

`TNGFileDescriptorFormat` class implements common FILEDESCRIPTOR data format and allows to drag/receive virtual files created on-the-fly from any data stream. NG Drag&Drop use special `TNGFileArray` type, which is a dynamic array of special `TNGFileDescriptor` records, for working with FILEDESCRIPTOR data. Please note that this format is specially designed to be used together with TNGFileContentsFormat format. While this format provide descriptions (like names, sizes, file dates and attributes) of dragging virtual files, TNGFileContentsFormat provides data streams for dragging virtual files.

`TNGFileDescriptor` record allows to specify the following file properties:

- `Name` - the name of file, like 'MyText.txt'
- `Clsid` - is the special GUID like identifier for special file system objects, like "Recycle Bin". Please read MSDN documentation.
- `Attributes` - file attributes, like for any other file.
- `CreationTime`, `LastAccessTime`, `LastWriteTime` - file times.
- `Size` - the size of corresponding data stream, in bytes.
- `Flags` - this property indicates, which other properties has been specified. A flag is set up automatically, when the corresponding property value is assigned. Actually, only `Name` property is required to be specified, all other properties are optional.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag virtual files at the source side, while `Ref` method can be used to receive files data data at the target side. NG Drag&Drop declares special type alias `CF.FILEDESCRIPTOR` to make user's code more readable:

```
SetLegnth(d, 1);
d[0].Name := 'c:\MyText1.txt';
d[0].Size := 10;

NGDropSource.Add(CF.FILEDESCRIPTOR.Data(d))
            .Add(CF.FILECONTENTS.Data(...))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  d:   TNGFileArray;
  cnt: TNGFileContents;
begin
  if C.Accept(CF.FILEDESCRIPTOR.Ref(@d)) and
     C.Accept(CF.FILECONTENTS.Ref(@cnt)) then
    ShowFiles(d);
end);
```

5.13 TNGFileContentsFormat

`TNGFileContentsFormat` class implements common `FILECONTENTS` data format and allows to drag/receive virtual files created on-the-fly from any data stream. NG Drag&Drop use `array of IStream` type at the source side, and a special `TNGFileContents` record, which allows to query `IStream` for each file, at the target side. Please note that this format is specially designed to be used together with TNGFileDescriptorFormat format. While TNGFileDescriptorFormat format provide descriptions (like names, sizes, file dates and attributes) of dragging virtual files, this format provides data streams for dragging virtual files. please also note, that `TNGFileContents` record has no `Count` property, since the count of dragging files should be determined from the corresponding TNGFileDescriptorFormat data.

The class declares two methods: `Data` and `Ref`. `Data` method can be used to drag virtual files at the source side, while `Ref` method can be used to receive files data data at the target side. NG Drag&Drop declares special type alias `CF.FILECONTENTS` to make user's code more readable:

```
SetLength(cnt, 1);
cnt[0] := TStreamAdapter.Create(MyFileStream, soOwned);

NGDropSource.Add(CF.FILEDESCRIPTOR.Data(...))
            .Add(CF.FILECONTENTS.Data(cnt))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  d:   TNGFileArray;
  cnt: TNGFileContents;
  s:   TStream;
begin
  if C.Accept(CF.FILEDESCRIPTOR.Ref(@d)) and
     C.Accept(CF.FILECONTENTS.Ref(@cnt)) then
  begin
    s := TOleStream.Create(cnt[0]);
    ShowFileContent(s);
    s.Free;
  end;
end);
```

`TStreamAapter` standard Delphi class can be used to convert any usual `TStream` to OLE `IStream`. Please note, that the resulting `IStream` object is referenced by OLE and its actually unknown, when it will be

released. So, its recommended to use `soOwned` value in the `TStreamAapter` constructor call to allow resulting `IStream` own initial `TStream` object.
To convert `IStream` back to `TStream` at the target side, `TOleStream` standard Delphi class can be used.

5.14 Custom Formats

NG Drag&Drop provides the ability to declare custom data formats. This feature can be used to implement application's private drag&drop formats or to implement missing commonly used formats.

An easy way to declare custom format based on some predefined format is to use `CustomFormatAttribute` attribute. A custom format should have its own unique name, which is used to register the format in the system:

```
type
  [CustomFormat('My Unique Format Name')]
  TMyFormat = class(TNGTextFormat);
```

That it! The custom format is declared. And so, it can be used like any other data format:

```
NGDropSource.Add(TMyFormat.Data('My dragging text'))
            .Execute;

NGDropTarget.Register(MyTargetPanel, procedure(C: TNGTargetContext)
var
  s: AnsiString;
begin
  if C.Accept(TMyFormat.Ref(@s)) then
    Edit1.Text := string(s);
end);
```

Advanced users can also implement custom formats descending the class directly from `TNGDataFormat` base class, just like built-in formats are implemented. This way the user have to deal with some low-level OLE drag&drop stuff, for example, `TStgMedium` and `TFormatEtc` WinAPI structures, which are out of current documentation scope. The source code in NG.DragDrop.Formats.pas unit, where all built-in formats are declared can be used as a reference implementation.