Delphi Generics.Collections

Table of Contents

1 Generics.Collections.TCollectionNotification

Type of change to collection for **OnNotify** event.

Description

```
Pascal
          TCollectionNotification = (cnAdded, cnRemoved, cnExtracted);
C++
        enum TCollectionNotification { cnAdded, cnRemoved, cnExtracted };
```

This table lists TCollectionNotification values.

Value	Meaning
cnAdded	Item added to collection.
cnRemoved	Item removed from collection.
cnExtracted	Item extracted from collection, i.e., removed and its value returned.

See Also

OnNotify (see page 85)

OnNotify (see page 139)

OnNotify (see page 157)

TCollectionNotifyEvent (see page 3)

2 Generics.Collections.TCollectionNotifyEvent

Event handler for **OnNotify** event.

Description

```
Pascal
          TCollectionNotifyEvent<T> = procedure(Sender: TObject; const Item: T;  Action:
TCollectionNotification) of object;
C++
         #define _decl_TCollectionNotifyEvent__1(T, _DECLNAME) void __fastcall (__closure
*_DECLNAME)(System::TObject* Sender, const T Item, TCollectionNotification Action);
```

TCollectionNotifyEvent is an event handler that can be set for an **OnNotify** event. This routine is called after the collection changes.

Sender is the collection object affected by the event. **Item** is an item that changed in the collection. **Action** is a TCollectionNotification that indicates the kind of change.

See Also

OnNotify (see page 85)

OnNotify (see page 139)

OnNotify (see page 157)

TCollectionNotification (see page 1)

3 Generics.Collections.TDictionary

Collection of key-value pairs.

Description

TDictionary represents a generic collection of key-value pairs.

This class provides a mapping from a collection of keys to a collection of values. When you create a TDictionary object, you can specify various combinations of initial capacity, equality operation, and initial content.

You can add a key that is associated with a corresponding value with the Add or AddOrSetValue methods. You can remove entries with Remove or Clear, which removes all key-value pairs. Adding or removing a key-value pair and looking up a key are efficient, close to O(1), because keys are hashed. A key must not be **nil** (though a value may be **nil**) and there must be an equality comparison operation for keys.

You can test for the presence or keys and values with the TryGetValue, ContainsKey and ContainsValue methods.

The Items property lists all Count dictionary entries. You can also set and get values by indexing the Items property. Setting the value this way overwrites any existing value.

The class TObjectDictionary inherits from TDictionary and provides an automatic mechanism for freeing objects removed from dictionary entries.

Delphi Examples:

```
{
This example requires a button and two TListboxes on a form.  The
Generics.Collections objects are created dynamically.  Notice that you can
call the TList Soft method with the TComparer as a parameter, or create the
TList with the TComparer and just call Sort.  Also, do not free reverseComp
after associating it with sortlist.
}
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Generics.Defaults,
    Generics.Collections;   // must appear after Classes to use the correct
TCollectionNotification

type
    TForm2 = class(TForm)
        Button1: TButton;
        ListBox1: TListBox;
```

```
        ListBox2: TListBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        ListBox3: TListBox;
        ListBox4: TListBox;
        Label4: TLabel;
        ListBox5: TListBox;
        Label5: TLabel;
        ListBox6: TListBox;
        Label6: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        Dictionary: TDictionary<Integer, Integer>;
        procedure OnDictAdd(
            Sender: TObject; const Item: Integer; Action: TCollectionNotification);
    end;
    TReverseIntComparer = class(TComparer<Integer>)
        function Compare(const Left, Right: Integer): Integer; override;
    end;

var
    Form2: TForm2;

implementation

{$R *.dfm}
{$APPTYPE CONSOLE}

function TReverseIntComparer.Compare(const Left, Right: Integer): Integer;
begin
    if Left < Right then
        Result := 1
    else if Left > Right then
        Result := -1
    else
        Result := 0;
end;

procedure TForm2.OnDictAdd(Sender: TObject; const Item: Integer; Action:
TCollectionNotification);
begin
    if Action = cnAdded then
        WriteLn('TDictionary Key has been added to the dictionary');
    if Action = cnRemoved then
        WriteLn('TDictionary Key has been removed from the dictionary');
end;

procedure TForm2.Button1Click(Sender: TObject);
var
    mylist, sortlist: TList<Integer>;
    reverseComp: TReverseIntComparer;
    myarray: TArray;
    values: array of Integer;
    i: Integer;
begin
    myarray:= TArray.Create;
    Dictionary.Add(0, 0);
    try
        Randomize;
        mylist:= Generics.Collections.TList<Integer>.Create;
        reverseComp:= TReverseIntComparer.Create;
```

```
        for i := 0 to 100 - 1 do
            mylist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox1.Items.Add(IntToStr(mylist[i]));
        mylist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox2.Items.Add(IntToStr(mylist[i]));
        mylist.Delete(0);
        mylist.Delete(2);
        mylist.Delete(4);
        for i := 0 to mylist.Count - 1 do
            Listbox3.Items.Add(IntToStr(mylist[i]));
        mylist.Sort(reverseComp);
        for i := 0 to mylist.Count - 1 do
            Listbox4.Items.Add(IntToStr(mylist[i]));
        sortlist:= Generics.Collections.TList<Integer>.Create(reverseComp);
        for i := 0 to 100 - 1 do
            sortlist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox5.Items.Add(IntToStr(sortlist[i]));
        sortlist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox6.Items.Add(IntToStr(mylist[i]));
    finally
//    reverseComp.Free;
        sortlist.Free;
        mylist.Free;
        Dictionary.Free;
        myarray.Free;
    end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
    Dictionary := TDictionary<Integer, Integer>.Create;
    Dictionary.OnKeyNotify := Form2.OnDictAdd;
end;
```

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    // Prints type info for string type
    TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
    GString->PrintTypeInfo(Memo1);

    // Prints type info for Byte type
    TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
    GByte->PrintTypeInfo(Memo1);

    // Prints type info for Double type
    TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
    GDouble->PrintTypeInfo(Memo1);

    // Prints type info for "array of String" type
    // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of
```

```
string>();
    // GStringArray->PrintTypeInfo(Memo1);
}
```

See Also

TObjectDictionary (see page 97)

Count (see page 19)

Items (see page 25)

Add (see page 9)

AddOrSetValue (see page 11)

Clear (see page 13)

ContainsKey (see page 15)

ContainsValue (see page 17)

Remove (see page 33)

TryGetValue (see page 37)

4 Generics.Collections.TDictionary.Add

Add key-value pair.

Description

```
Pascal
            procedure Add(const Key: TKey; const Value: TValue);
C++
          __fastcall Add(const TKey Key, const TValue Value);
```

Add adds a key and its corresponding value to the dictionary. The key cannot be **nil**, but the value can.

If the key already exists in the dictionary, an exception is thrown.

An OnKeyNotify event and an OnValueNotify event occur indicating an entry was added to the dictionary.

The Items property lists all dictionary entries. You can also set and get values by indexing the Items property directly. For instance, you can set a value this way:

```
Items[key] := value;
```

Setting the value this way overwrites the value for an existing key, but does not raise an exception.

See Also

Items (see page 25)

AddOrSetValue (see page 11)

OnKeyNotify (see page 27)

OnValueNotify (see page 31)

5 Generics.Collections.TDictionary.AddOrSetValue

Add key-value pair even when key already exists.

Description

```
Pascal
          procedure AddOrSetValue(const Key: TKey; const Value: TValue);
C++
        __fastcall AddOrSetValue(const TKey Key, const TValue Value);
```

AddOrSetValue adds a key-value pair to a dictionary even if the key already exists. The key cannot be **nil**, but the value can. This method checks to see if the key exists in the dictionary, and if it does, it is equivalent to `Items[key] := value;`. Otherwise it is equivalent to `Add(key, value);`.

An OnKeyNotify event and an OnValueNotify event occur indicating an entry was added to the dictionary.

See Also

Items (see page 25)

Add (see page 9)

OnKeyNotify (see page 27)

OnValueNotify (see page 31)

6 Generics.Collections.TDictionary.Clear

Clear all data.

Description

```
Pascal
            procedure Clear;
C++
          __fastcall Clear();
```

Clear removes all keys and values from the dictionary. The Count property is set to 0. The capacity is also set to 0. This operation requires O(n) time, where n is Count, the number of dictionary entries.

Note: Clear does not free the items as they are removed. If you need to free them, use the OnKeyNotify event and the OnValueNotify event, which occur for every entry removed and provides the removed items.

See Also

Count (see page 19)

Remove (see page 33)

TrimExcess (see page 35)

OnKeyNotify (see page 27)

OnValueNotify (see page 31)

7 Generics.Collections.TDictionary.ContainsKey

Test if key in dictionary.

Description

```
Pascal
          function ContainsKey(const Key: TKey): Boolean;
C++
        bool __fastcall ContainsKey(const TKey Key);
```

ContainsKey returns true if the given key is in the dictionary and false otherwise. This is an O(1) operation.

See Also

AddOrSetValue (see page 11)

ContainsValue (see page 17)

TryGetValue (see page 37)

8 Generics.Collections.TDictionary.ContainsValue

Check if value in dictionary.

Description

Pascal

```
function ContainsValue(const Value: TValue): Boolean;
```

C++

```
bool __fastcall ContainsValue(const TValue Value);
```

ContainsValue returns true if the given value is in the dictionary and false otherwise. This is an O(n) operation, where n is the number of entries in the Count property.

See Also

Count (see page 19)

ContainsKey (see page 15)

9 Generics.Collections.TDictionary.Count

Number of entries.

Description

```
Pascal
            property Count: Integer;
C++
          __property int Count;
```

Count holds the number of key-value pairs in the dictionary. The Items property holds Count entries.

See Also

Items (see page 25)

10
Generics.Collections.TDictionary.Create

Create dictionary.

Description

```
Pascal
            constructor Create(ACapacity: Integer = 0); overload;
            constructor Create(const AComparer: IEqualityComparer<TKey>); overload;
            constructor Create(ACapacity: Integer; const AComparer: IEqualityComparer<TKey>);
overload;
            constructor Create(Collection: TEnumerable<TPair<TKey,TValue>>); overload;
            constructor Create(Collection: TEnumerable<TPair<TKey,TValue>>; const AComparer:
IEqualityComparer<TKey>); overload;
C++
          __fastcall TDictionary__2(int ACapacity)/* overload */;
          __fastcall TDictionary__2(const
System::DelphiInterface<Generics_defaults::IEqualityComparer__1<TKey> >  AComparer)/* overload
*/;
          __fastcall TDictionary__2(int ACapacity, const
System::DelphiInterface<Generics_defaults::IEqualityComparer__1<TKey> >  AComparer)/* overload
*/;
          __fastcall TDictionary__2(TEnumerable__1<TPair__2<TKey,TValue> >* Collection)/*
overload */;
          __fastcall TDictionary__2(TEnumerable__1<TPair__2<TKey,TValue> >* Collection, const
System::DelphiInterface<Generics_defaults::IEqualityComparer__1<TKey> >  AComparer)/* overload
*/;
```

This overloaded method creates and initializes a dictionary instance. Various combinations of parameters may be used to specify the initial capacity **ACapacity**, an equality comparison function **AComparer**, or an initial collection of key-value items **Collection**.

See Also

Destroy (see page 23)

11
Generics.Collections.TDictionary.Destroy

Destroy dictionary.

Description

```
Pascal
            destructor Destroy; override;
C++
          __fastcall virtual ~TDictionary__2();
```

This method destroys an instance of a dictionary using Clear.

Note: Clear does not free the items as they are removed. If you need to free them, use the OnNotify event, which occurs for every entry removed and provides the removed items.

See Also

Clear (see page 13)

Create (see page 21)

12
Generics.Collections.TDictionary.Items

Indexable list of all dictionary entries.

Description

```
Pascal
          property Items[const Key: TKey]: TValue;
C++
        __property TValue Items[const TKey Key];
```

Items is an indexable list of all key-value pairs in the dictionary.

The Count property holds the number of dictionary entries in Items.

You can set and get values by indexing the Items property. Setting the value this way overwrites an existing value and does not raise an exception.

See Also

Count (see page 19)

Add (see page 9)

AddOrSetValue (see page 11)

13
Generics.Collections.TDictionary.OnKeyNotify

Occurs when a dictionary key pair changes.

Description

```
Pascal
            property OnNotify: TCollectionNotifyEvent<TKey>;
C++
          __property _decl_TCollectionNotifyEvent__1(TKey, OnKeyNotify);
```

The OnKeyNotify event occurs when items are added or removed from the dictionary. Multiple events may occur for a single operation. This allows removed objects to be freed.

Delphi Examples:

```
{
This example requires a button and two TListboxes on a form.  The
Generics.Collections objects are created dynamically.  Notice that you can
call the TList Soft method with the TComparer as a parameter, or create the
TList with the TComparer and just call Sort.  Also, do not free reverseComp
after associating it with sortlist.
}
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Generics.Defaults,
    Generics.Collections;   // must appear after Classes to use the correct
TCollectionNotification

type
    TForm2 = class(TForm)
        Button1: TButton;
        ListBox1: TListBox;
        ListBox2: TListBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        ListBox3: TListBox;
```

```
        ListBox4: TListBox;
        Label4: TLabel;
        ListBox5: TListBox;
        Label5: TLabel;
        ListBox6: TListBox;
        Label6: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        Dictionary: TDictionary<Integer, Integer>;
        procedure OnDictAdd(
            Sender: TObject; const Item: Integer; Action: TCollectionNotification);
    end;
    TReverseIntComparer = class(TComparer<Integer>)
        function Compare(const Left, Right: Integer): Integer; override;
    end;

var
    Form2: TForm2;

implementation

{$R *.dfm}
{$APPTYPE CONSOLE}

function TReverseIntComparer.Compare(const Left, Right: Integer): Integer;
begin
    if Left < Right then
        Result := 1
    else if Left > Right then
        Result := -1
    else
        Result := 0;
end;

procedure TForm2.OnDictAdd(Sender: TObject; const Item: Integer; Action:
TCollectionNotification);
begin
    if Action = cnAdded then
        WriteLn('TDictionary Key has been added to the dictionary');
    if Action = cnRemoved then
        WriteLn('TDictionary Key has been removed from the dictionary');
end;

procedure TForm2.Button1Click(Sender: TObject);
var
    mylist, sortlist: TList<Integer>;
    reverseComp: TReverseIntComparer;
    myarray: TArray;
    values: array of Integer;
    i: Integer;
begin
    myarray:= TArray.Create;
    Dictionary.Add(0, 0);
    try
        Randomize;
        mylist:= Generics.Collections.TList<Integer>.Create;
        reverseComp:= TReverseIntComparer.Create;
        for i := 0 to 100 - 1 do
            mylist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox1.Items.Add(IntToStr(mylist[i]));
        mylist.Sort;
```

```
        for i := 0 to mylist.Count - 1 do
            Listbox2.Items.Add(IntToStr(mylist[i]));
        mylist.Delete(0);
        mylist.Delete(2);
        mylist.Delete(4);
        for i := 0 to mylist.Count - 1 do
            Listbox3.Items.Add(IntToStr(mylist[i]));
        mylist.Sort(reverseComp);
        for i := 0 to mylist.Count - 1 do
            Listbox4.Items.Add(IntToStr(mylist[i]));
        sortlist:= Generics.Collections.TList<Integer>.Create(reverseComp);
        for i := 0 to 100 - 1 do
            sortlist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox5.Items.Add(IntToStr(sortlist[i]));
        sortlist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox6.Items.Add(IntToStr(mylist[i]));
    finally
//    reverseComp.Free;
        sortlist.Free;
        mylist.Free;
        Dictionary.Free;
        myarray.Free;
    end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
    Dictionary := TDictionary<Integer, Integer>.Create;
    Dictionary.OnKeyNotify := Form2.OnDictAdd;
end;
```

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    // Prints type info for string type
    TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
    GString->PrintTypeInfo(Memo1);

    // Prints type info for Byte type
    TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
    GByte->PrintTypeInfo(Memo1);

    // Prints type info for Double type
    TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
    GDouble->PrintTypeInfo(Memo1);

    // Prints type info for "array of String" type
    // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of
string>();
    // GStringArray->PrintTypeInfo(Memo1);
}
```

See Also

Add (see page 9)

AddOrSetValue (see page 11)

Clear (see page 13)

Remove (see page 33)

TCollectionNotifyEvent (see page 3)

TCollectionNotification (see page 1)

14
Generics.Collections.TDictionary.OnValueNotify

Occurs when a dictionary key pair changes.

Description

```
Pascal
          property OnNotify: TCollectionNotifyEvent<TValue>;
C++
        __property _decl_TCollectionNotifyEvent__1(TValue, OnValueNotify);
```

The OnValueNotify event occurs when items are added or removed from the dictionary. Multiple events may occur for a single operation. This allows removed objects to be freed.

See Also

Add (see page 9)

AddOrSetValue (see page 11)

Clear (see page 13)

Remove (see page 33)

TCollectionNotifyEvent (see page 3)

TCollectionNotification (see page 1)

15
Generics.Collections.TDictionary.Remove

Remove key-value pair.

Description

Pascal
```
procedure Remove(const Key: TKey);
```
C++
```
__fastcall Remove(const TKey Key);
```

Remove removes the given key and its associated value from the dictionary. No exception is thrown if the key is not in the dictionary. This is an O(1) operation.

An OnKeyNotify event and an OnValueNotify event occur indicating an entry was removed from the dictionary.

See Also

Destroy (see page 23)

Clear (see page 13)

TrimExcess (see page 35)

OnKeyNotify (see page 27)

OnValueNotify (see page 31)

16 Generics.Collections.TDictionary.TrimExcess

Reduce capacity to current number of entries.

Description

```
Pascal
          procedure TrimExcess;
C++
        __fastcall TrimExcess();
```

TrimExcess changes the capacity to the number of dictionary entries, held in Count.

This method rehashes the internal hash table to save space. This is only useful after a lot of items have been deleted from the dictionary.

See Also

Count (see page 19)

Remove (see page 33)

Clear (see page 13)

TrimExcess (see page 95)

TrimExcess (see page 143)

TrimExcess (see page 165)

17
Generics.Collections.TDictionary.TryGetValue

Try to get value for key.

Description

```
Pascal
          function TryGetValue(const Key: TKey; out Value: TValue): Boolean;
C++
        bool __fastcall TryGetValue(const TKey Key, /* out */ TValue &Value);
```

TryGetValue returns true if the given key is in the dictionary and provides its value in **Value**. Otherwise, it returns false and **Value** is set to the default value type of **TValue**. No exception is raised if the key is not in the dictionary. This is an O(1) operation.

See Also

ContainsKey (see page 15)

18 Generics.Collections.TDictionaryOwnerships

Set of ownerships for TObjectDictionary.

Description

```
Pascal
          TDictionaryOwnerships = set of (doOwnsKeys, doOwnsValues); test
C++
        typedef Set<Generics_collections__91, doOwnsKeys, doOwnsValues>
TDictionaryOwnerships;
```

TDictionaryOwnerships is a set of ownerships for TObjectDictionary objects specified at object creation. None, one or both may be specified. If the dictionary owns the key and/or value, the key and/or value is freed when the entry is removed from the dictionary.

This table lists TDictionaryOwnerships values.

Value	Meaning
doOwnsKeys	Dictionary owns keys in entries.
doOwnsValues	Dictionary owns values in entries.

See Also

Create (see page 99)

19 Generics.Collections.TList

Ordered list.

Description

TList represents an ordered list, accessible by an index.

You can create a list with a specific collection of items and a comparison operator.

You can add, change, insert or remove an item from a list, or clear the entire list. You can add **nil** objects to the list.

You can sort, search and reverse a list.

Count contains the number of items in the queue. Capacity is the number of items the list can hold before being resized. You can also set and get values by indexing the Items array.

An OnNotify event tells you when the list has changed.

The class TObjectList inherits from TList and provides an automatic mechanism for freeing objects removed from lists.

Delphi Examples:

```
{
This example requires a button and two TListboxes on a form.  The
Generics.Collections objects are created dynamically.  Notice that you can
call the TList Soft method with the TComparer as a parameter, or create the
TList with the TComparer and just call Sort.  Also, do not free reverseComp
after associating it with sortlist.
}
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Generics.Defaults,
    Generics.Collections;   // must appear after Classes to use the correct
TCollectionNotification

type
    TForm2 = class(TForm)
        Button1: TButton;
        ListBox1: TListBox;
        ListBox2: TListBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        ListBox3: TListBox;
```

```
        ListBox4: TListBox;
        Label4: TLabel;
        ListBox5: TListBox;
        Label5: TLabel;
        ListBox6: TListBox;
        Label6: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        Dictionary: TDictionary<Integer, Integer>;
        procedure OnDictAdd(
            Sender: TObject; const Item: Integer; Action: TCollectionNotification);
    end;
    TReverseIntComparer = class(TComparer<Integer>)
        function Compare(const Left, Right: Integer): Integer; override;
    end;

var
    Form2: TForm2;

implementation

{$R *.dfm}
{$APPTYPE CONSOLE}

function TReverseIntComparer.Compare(const Left, Right: Integer): Integer;
begin
    if Left < Right then
        Result := 1
    else if Left > Right then
        Result := -1
    else
        Result := 0;
end;

procedure TForm2.OnDictAdd(Sender: TObject; const Item: Integer; Action:
TCollectionNotification);
begin
    if Action = cnAdded then
        WriteLn('TDictionary Key has been added to the dictionary');
    if Action = cnRemoved then
        WriteLn('TDictionary Key has been removed from the dictionary');
end;

procedure TForm2.Button1Click(Sender: TObject);
var
    mylist, sortlist: TList<Integer>;
    reverseComp: TReverseIntComparer;
    myarray: TArray;
    values: array of Integer;
    i: Integer;
begin
    myarray:= TArray.Create;
    Dictionary.Add(0, 0);
    try
        Randomize;
        mylist:= Generics.Collections.TList<Integer>.Create;
        reverseComp:= TReverseIntComparer.Create;
        for i := 0 to 100 - 1 do
            mylist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox1.Items.Add(IntToStr(mylist[i]));
        mylist.Sort;
```

```
        for i := 0 to mylist.Count - 1 do
            Listbox2.Items.Add(IntToStr(mylist[i]));
        mylist.Delete(0);
        mylist.Delete(2);
        mylist.Delete(4);
        for i := 0 to mylist.Count - 1 do
            Listbox3.Items.Add(IntToStr(mylist[i]));
        mylist.Sort(reverseComp);
        for i := 0 to mylist.Count - 1 do
            Listbox4.Items.Add(IntToStr(mylist[i]));
        sortlist:= Generics.Collections.TList<Integer>.Create(reverseComp);
        for i := 0 to 100 - 1 do
            sortlist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox5.Items.Add(IntToStr(sortlist[i]));
        sortlist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox6.Items.Add(IntToStr(mylist[i]));
    finally
//    reverseComp.Free;
        sortlist.Free;
        mylist.Free;
        Dictionary.Free;
        myarray.Free;
    end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
    Dictionary := TDictionary<Integer, Integer>.Create;
    Dictionary.OnKeyNotify := Form2.OnDictAdd;
end;
```

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    // Prints type info for string type
    TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
    GString->PrintTypeInfo(Memo1);

    // Prints type info for Byte type
    TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
    GByte->PrintTypeInfo(Memo1);

    // Prints type info for Double type
    TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
    GDouble->PrintTypeInfo(Memo1);

    // Prints type info for "array of String" type
    // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of
string>();
    // GStringArray->PrintTypeInfo(Memo1);
}
```

See Also

TObjectList (see page 101)

Capacity (see page 53)

Count (see page 59)

Items (see page 81)

Add (see page 45)

AddRange (see page 49)

BinarySearch (see page 51)

Clear (see page 55)

Create (see page 61)

Delete (see page 65)

DeleteRange (see page 69)

IndexOf (see page 75)

Insert (see page 77)

InsertRange (see page 79)

Remove (see page 87)

Reverse (see page 89)

Sort (see page 91)

OnNotify (see page 85)

20 Generics.Collections.TList.Add

Add item to end of list.

Description

```
Pascal
            function Add(const Value: T): Integer;
C++
          int __fastcall Add(const T Value);
```

Add adds a given item to the end of the list. You can add **nil**. The capacity, Capacity, of the list is increased if necessary. This is an O(1) operation.

An OnNotify event occurs indicating an entry was added to the list.

Delphi Examples:

```
{
This example requires a button and two TListboxes on a form.  The
Generics.Collections objects are created dynamically.  Notice that you can
call the TList Soft method with the TComparer as a parameter, or create the
TList with the TComparer and just call Sort.  Also, do not free reverseComp
after associating it with sortlist.
}
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Generics.Defaults,
    Generics.Collections;   // must appear after Classes to use the correct
TCollectionNotification

type
    TForm2 = class(TForm)
        Button1: TButton;
        ListBox1: TListBox;
        ListBox2: TListBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        ListBox3: TListBox;
        ListBox4: TListBox;
        Label4: TLabel;
        ListBox5: TListBox;
        Label5: TLabel;
        ListBox6: TListBox;
```

```
        Label6: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        Dictionary: TDictionary<Integer, Integer>;
        procedure OnDictAdd(
            Sender: TObject; const Item: Integer; Action: TCollectionNotification);
    end;
    TReverseIntComparer = class(TComparer<Integer>)
        function Compare(const Left, Right: Integer): Integer; override;
    end;

var
    Form2: TForm2;

implementation

{$R *.dfm}
{$APPTYPE CONSOLE}


function TReverseIntComparer.Compare(const Left, Right: Integer): Integer;
begin
    if Left < Right then
        Result := 1
    else if Left > Right then
        Result := -1
    else
        Result := 0;
end;

procedure TForm2.OnDictAdd(Sender: TObject; const Item: Integer; Action:
TCollectionNotification);
begin
    if Action = cnAdded then
        WriteLn('TDictionary Key has been added to the dictionary');
    if Action = cnRemoved then
        WriteLn('TDictionary Key has been removed from the dictionary');
end;

procedure TForm2.Button1Click(Sender: TObject);
var
    mylist, sortlist: TList<Integer>;
    reverseComp: TReverseIntComparer;
    myarray: TArray;
    values: array of Integer;
    i: Integer;
begin
    myarray:= TArray.Create;
    Dictionary.Add(0, 0);
    try
        Randomize;
        mylist:= Generics.Collections.TList<Integer>.Create;
        reverseComp:= TReverseIntComparer.Create;
        for i := 0 to 100 - 1 do
            mylist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox1.Items.Add(IntToStr(mylist[i]));
        mylist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox2.Items.Add(IntToStr(mylist[i]));
        mylist.Delete(0);
        mylist.Delete(2);
```

```
        mylist.Delete(4);
        for i := 0 to mylist.Count - 1 do
            Listbox3.Items.Add(IntToStr(mylist[i]));
        mylist.Sort(reverseComp);
        for i := 0 to mylist.Count - 1 do
            Listbox4.Items.Add(IntToStr(mylist[i]));
        sortlist:= Generics.Collections.TList<Integer>.Create(reverseComp);
        for i := 0 to 100 - 1 do
            sortlist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox5.Items.Add(IntToStr(sortlist[i]));
        sortlist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox6.Items.Add(IntToStr(mylist[i]));
    finally
//    reverseComp.Free;
        sortlist.Free;
        mylist.Free;
        Dictionary.Free;
        myarray.Free;
    end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
    Dictionary := TDictionary<Integer, Integer>.Create;
    Dictionary.OnKeyNotify := Form2.OnDictAdd;
end;
```

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    // Prints type info for string type
    TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
    GString->PrintTypeInfo(Memo1);

    // Prints type info for Byte type
    TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
    GByte->PrintTypeInfo(Memo1);

    // Prints type info for Double type
    TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
    GDouble->PrintTypeInfo(Memo1);

    // Prints type info for "array of String" type
    // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of
string>();
    // GStringArray->PrintTypeInfo(Memo1);
}
```

See Also

Capacity (see page 53)

AddRange (see page 49)

OnNotify (see page 85)

21 Generics.Collections.TList.AddRange

Add collection to end of list.

Description

```
Pascal
            procedure AddRange(const Values: array of T); overload;
            procedure AddRange(const Collection: IEnumerable<T>); overload;
            procedure AddRange(Collection: TEnumerable<T>); overload;
C++
          void __fastcall AddRange(T const *Values, const int Values_Size)/* overload */;
          void __fastcall AddRange(const System::DelphiInterface<System::IEnumerable__1<T> >
Collection)/* overload */;
          void __fastcall AddRange(TEnumerable__1<T>* Collection)/* overload */;
```

AddRange adds a collection of items to the end of a list. The capacity, Capacity, of the list is increased if necessary. This is an O(n) operation where n = number of elements in added collection.

An OnNotify event occurs indicating entries were added to the list.

See Also

Capacity (see page 53)

Add (see page 45)

InsertRange (see page 79)

OnNotify (see page 85)

22
Generics.Collections.TList.BinarySearch

Search sorted list for element using binary search.

Description

```
Pascal
            function BinarySearch(const Item: T; out Index: Integer): Boolean; overload;
            function BinarySearch(const Item: T; out Index: Integer; const AComparer:
IComparer<T>): Boolean; overload;
C++
          bool __fastcall BinarySearch(const T Item, /* out */ int &Index)/* overload */;
          bool __fastcall BinarySearch(const T Item, /* out */ int &Index, const
System::DelphiInterface<Generics_defaults::IComparer__1<T> >  AComparer)/* overload */;
```

The overloaded method BinarySearch searches for the list element **Item** using a binary search. The method returns true if it finds the element and false otherwise. If found, **Index** contains the zero-based index of **Item**. If not found, **Index** contains the index of the first entry larger than **Item**.

Note: BinarySearch requires that the list be sorted. The IndexOf method does not require a sorted list, but is usually slower than BinarySearch.

If there is more than one element in the list equal to **Item**, the index of the first match is returned in **Index**. This is the index of any of the matching items, not necessarily the first.

A comparison function **AComparer** may be provided to compare elements.

If **Item** is out of the range of the list, an **EArgumentOutOfRangeException** exception is raised.

This is an O(log n) operation for a list with n entries.

See Also

IndexOf (see page 75)

Sort (see page 91)

23 Generics.Collections.TList.Capacity

List capacity.

Description

```
Pascal
            property Capacity: Integer;
C++
          __property int Capacity;
```

Capacity gets or sets the list capacity, that is, the maximum size of the list without resizing. The capacity cannot be set to less than Count, the actual number of items in the list.

The TrimExcess method reduces the capacity of the list to its current number of elements, Count.

See Also

Count (see page 59)

Items (see page 81)

TrimExcess (see page 95)

24 Generics.Collections.TList.Clear

Remove all list entries.

Description

```
Pascal
            procedure Clear;
C++
          __fastcall Clear();
```

Clear removes all entries from a list. This is a O(1) operation. Both Capacity and Count are set to 0.

Note: Clear does not free the items as they are removed. If you need to free them, use the OnNotify event, which occurs for every item removed and provides the removed item.

See Also

Capacity (see page 53)

Count (see page 59)

Remove (see page 87)

OnNotify (see page 85)

25 Generics.Collections.TList.Contains

Test if element in list.

Description

Pascal

```
function Contains(const Value: T): Boolean;
```

C++

```
bool __fastcall Contains(const T Value);
```

Contains returns true if the element **Value** is in the list and false otherwise. This method only indicates whether the element is in the list or not; use IndexOf to get the index of an element in a list.

Since the search for the element is linear, it is an O(n) operation for a list with n entries.

See Also

IndexOf (see page 75)

26 Generics.Collections.TList.Count

Number of list elements.

Description

```
Pascal
            property Count: Integer;
C++
          property int Count;
```

Count gets or sets the actual number of elements in the list. Count is always less than or equal to Capacity.

If an operation, such as Insert, would increase Count to be greater than Capacity, the list is automatically resized and Capacity is increased.

If Count is reduced, then Count - **Value** items at the end of the list are removed.

See Also

Capacity (see page 53)

Items (see page 81)

Insert (see page 77)

27 Generics.Collections.TList.Create

Create and initialize list instance.

Description

```
Pascal
            constructor Create; overload;
            constructor Create(const AComparer: IComparer<T>); overload;
            constructor Create(Collection: TEnumerable<T>); overload;
C++
          __fastcall TList__1(void)/* overload */;
          __fastcall TList__1(const System::DelphiInterface<Generics_defaults::IComparer__1<T>
>  AComparer)/* overload */;
          __fastcall TList__1(TEnumerable__1<T>* Collection)/* overload */;
```

This overloaded method creates and initializes a list instance.

AComparer is a comparison function. If not provided, the default comparator for the type is used.

Collection is a collection with which to initialize the list. The objects are added in the same order as in **Collection**. If **Collection** is specified, the creation is an O(n) operation, where n is the number of items in **Collection**.

Delphi Examples:

```
{
This example requires a button and two TListboxes on a form.  The
Generics.Collections objects are created dynamically.  Notice that you can
call the TList Soft method with the TComparer as a parameter, or create the
TList with the TComparer and just call Sort.  Also, do not free reverseComp
after associating it with sortlist.
}
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Generics.Defaults,
    Generics.Collections;   // must appear after Classes to use the correct
TCollectionNotification

type
    TForm2 = class(TForm)
        Button1: TButton;
        ListBox1: TListBox;
        ListBox2: TListBox;
        Label1: TLabel;
        Label2: TLabel;
```

```
        Label3: TLabel;
        ListBox3: TListBox;
        ListBox4: TListBox;
        Label4: TLabel;
        ListBox5: TListBox;
        Label5: TLabel;
        ListBox6: TListBox;
        Label6: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        Dictionary: TDictionary<Integer, Integer>;
        procedure OnDictAdd(
            Sender: TObject; const Item: Integer; Action: TCollectionNotification);
    end;
    TReverseIntComparer = class(TComparer<Integer>)
        function Compare(const Left, Right: Integer): Integer; override;
    end;

var
    Form2: TForm2;

implementation

{$R *.dfm}
{$APPTYPE CONSOLE}

function TReverseIntComparer.Compare(const Left, Right: Integer): Integer;
begin
    if Left < Right then
        Result := 1
    else if Left > Right then
        Result := -1
    else
        Result := 0;
end;

procedure TForm2.OnDictAdd(Sender: TObject; const Item: Integer; Action:
TCollectionNotification);
begin
    if Action = cnAdded then
        WriteLn('TDictionary Key has been added to the dictionary');
    if Action = cnRemoved then
        WriteLn('TDictionary Key has been removed from the dictionary');
end;

procedure TForm2.Button1Click(Sender: TObject);
var
    mylist, sortlist: TList<Integer>;
    reverseComp: TReverseIntComparer;
    myarray: TArray;
    values: array of Integer;
    i: Integer;
begin
    myarray:= TArray.Create;
    Dictionary.Add(0, 0);
    try
        Randomize;
        mylist:= Generics.Collections.TList<Integer>.Create;
        reverseComp:= TReverseIntComparer.Create;
        for i := 0 to 100 - 1 do
            mylist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
```

```
            Listbox1.Items.Add(IntToStr(mylist[i]));
        mylist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox2.Items.Add(IntToStr(mylist[i]));
        mylist.Delete(0);
        mylist.Delete(2);
        mylist.Delete(4);
        for i := 0 to mylist.Count - 1 do
            Listbox3.Items.Add(IntToStr(mylist[i]));
        mylist.Sort(reverseComp);
        for i := 0 to mylist.Count - 1 do
            Listbox4.Items.Add(IntToStr(mylist[i]));
        sortlist:= Generics.Collections.TList<Integer>.Create(reverseComp);
        for i := 0 to 100 - 1 do
            sortlist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox5.Items.Add(IntToStr(sortlist[i]));
        sortlist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox6.Items.Add(IntToStr(mylist[i]));
    finally
    //    reverseComp.Free;
        sortlist.Free;
        mylist.Free;
        Dictionary.Free;
        myarray.Free;
    end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
    Dictionary := TDictionary<Integer, Integer>.Create;
    Dictionary.OnKeyNotify := Form2.OnDictAdd;
end;
```

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    // Prints type info for string type
    TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
    GString->PrintTypeInfo(Memo1);

    // Prints type info for Byte type
    TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
    GByte->PrintTypeInfo(Memo1);

    // Prints type info for Double type
    TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
    GDouble->PrintTypeInfo(Memo1);

    // Prints type info for "array of String" type
    // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of
string>();
    // GStringArray->PrintTypeInfo(Memo1);
}
```

See Also

Destroy (see page 71)

InsertRange (see page 79)

28 Generics.Collections.TList.Delete

Remove entry at index.

Description

```
Pascal
            procedure Delete(Index: Integer);
C++
          __fastcall Delete(int Index);
```

Delete removes the list entry at the given index **Index**.

If **Index** is not valid for the list, an EArgumentOutOfRangeException exception is raised.

An OnNotify event occurs indicating an entry was removed from the list.

Delphi Examples:

```
{
This example requires a button and two TListboxes on a form.  The
Generics.Collections objects are created dynamically.  Notice that you can
call the TList Soft method with the TComparer as a parameter, or create the
TList with the TComparer and just call Sort.  Also, do not free reverseComp
after associating it with sortlist.
}
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Generics.Defaults,
    Generics.Collections;   // must appear after Classes to use the correct
TCollectionNotification

type
    TForm2 = class(TForm)
        Button1: TButton;
        ListBox1: TListBox;
        ListBox2: TListBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        ListBox3: TListBox;
        ListBox4: TListBox;
        Label4: TLabel;
        ListBox5: TListBox;
        Label5: TLabel;
```

```
        ListBox6: TListBox;
        Label6: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        Dictionary: TDictionary<Integer, Integer>;
        procedure OnDictAdd(
            Sender: TObject; const Item: Integer; Action: TCollectionNotification);
    end;
    TReverseIntComparer = class(TComparer<Integer>)
        function Compare(const Left, Right: Integer): Integer; override;
    end;

var
    Form2: TForm2;

implementation

{$R *.dfm}
{$APPTYPE CONSOLE}

function TReverseIntComparer.Compare(const Left, Right: Integer): Integer;
begin
    if Left < Right then
        Result := 1
    else if Left > Right then
        Result := -1
    else
        Result := 0;
end;

procedure TForm2.OnDictAdd(Sender: TObject; const Item: Integer; Action:
TCollectionNotification);
begin
    if Action = cnAdded then
        WriteLn('TDictionary Key has been added to the dictionary');
    if Action = cnRemoved then
        WriteLn('TDictionary Key has been removed from the dictionary');
end;

procedure TForm2.Button1Click(Sender: TObject);
var
    mylist, sortlist: TList<Integer>;
    reverseComp: TReverseIntComparer;
    myarray: TArray;
    values: array of Integer;
    i: Integer;
begin
    myarray:= TArray.Create;
    Dictionary.Add(0, 0);
    try
        Randomize;
        mylist:= Generics.Collections.TList<Integer>.Create;
        reverseComp:= TReverseIntComparer.Create;
        for i := 0 to 100 - 1 do
            mylist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox1.Items.Add(IntToStr(mylist[i]));
        mylist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox2.Items.Add(IntToStr(mylist[i]));
        mylist.Delete(0);
        mylist.Delete(2);
```

```
        mylist.Delete(4);
        for i := 0 to mylist.Count - 1 do
            Listbox3.Items.Add(IntToStr(mylist[i]));
        mylist.Sort(reverseComp);
        for i := 0 to mylist.Count - 1 do
            Listbox4.Items.Add(IntToStr(mylist[i]));
            sortlist:= Generics.Collections.TList<Integer>.Create(reverseComp);
        for i := 0 to 100 - 1 do
            sortlist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox5.Items.Add(IntToStr(sortlist[i]));
        sortlist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox6.Items.Add(IntToStr(mylist[i]));
    finally
//    reverseComp.Free;
        sortlist.Free;
        mylist.Free;
        Dictionary.Free;
        myarray.Free;
    end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
    Dictionary := TDictionary<Integer, Integer>.Create;
    Dictionary.OnKeyNotify := Form2.OnDictAdd;
end;
```

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    // Prints type info for string type
    TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
    GString->PrintTypeInfo(Memo1);

    // Prints type info for Byte type
    TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
    GByte->PrintTypeInfo(Memo1);

    // Prints type info for Double type
    TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
    GDouble->PrintTypeInfo(Memo1);

    // Prints type info for "array of String" type
    // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of
string>();
    // GStringArray->PrintTypeInfo(Memo1);
}
```

See Also

Clear (see page 55)

DeleteRange (see page 69)

Extract (see page 73)

Insert (see page 77)

InsertRange (see page 79)

Remove (see page 87)

OnNotify (see page 85)

29
Generics.Collections.TList.DeleteRange

Remove several entries at index.

Description

```
Pascal
          procedure DeleteRange(AIndex, ACount: Integer);
C++
        __fastcall DeleteRange(int AIndex, int ACount);
```

DeleteRange removes **ACount** list entries at the given index **AIndex**.

If **AIndex** and **ACount** describe an invalid range for the list, an EArgumentOutOfRangeException exception is raised.

This is an O(ACount) operation.

An OnNotify event occurs indicating entries were removed from the list.

See Also

Clear (see page 55)

Delete (see page 65)

Insert (see page 77)

InsertRange (see page 79)

Remove (see page 87)

OnNotify (see page 85)

30 Generics.Collections.TList.Destroy

Destroy list.

Description

```
Pascal
            destructor Destroy; override;
C++
          __fastcall virtual ~TList__1();
```

This method destroys a list.

See Also

Create (see page 61)

31 Generics.Collections.TList.Extract

Remove and return list entry.

Description

```
Pascal
            function Extract(const Value: T): T;
C++
          T __fastcall Extract(const T Value);
```

Extract removes the entry **Value** from the list, returning this value. If **Value** is not in the list, it returns the default value of its type **T**.

An OnNotify event occurs indicating an entry was removed from the list. Extract is similar to Remove except for the event code indicating an element was extracted rather than removed and is provided so items may be removed without freeing.

See Also

Clear (see page 55)

Delete (see page 65)

DeleteRange (see page 69)

Remove (see page 87)

OnNotify (see page 85)

32 Generics.Collections.TList.IndexOf

Search for element using linear search.

Description

Pascal

```
function IndexOf(const Value: T): Integer;
```

C++

```
int __fastcall IndexOf(const T Value);
```

IndexOf searches for the list element **Value** using a linear search. The method returns the zero-based index of the first entry equal to **Value**. If not found, it returns -1.

Since the search is linear, it is an O(n) operation for a list with n entries. The method BinarySearch is usually faster, but requires a sorted list.

See Also

BinarySearch (see page 51)

Contains (see page 57)

LastIndexOf (see page 83)

33 Generics.Collections.TList.Insert

Insert entry in list.

Description

Pascal

```
procedure Insert(Index: Integer; const Value: T);
```

C++

```
__fastcall Insert(int Index, const T Value);
```

Insert inserts an element **Value** in the list at the index **Index**. If the list **Count** is already equal to **Capacity**, **Capacity** is increased.

If **Index** is not valid for the list, an **EArgumentOutOfRangeException** exception is raised.

This is an O(n) operation, where n is the number of items in the list.

An OnNotify event occurs indicating an item was inserted in the list.

See Also

Count (see page 59)

Capacity (see page 53)

Delete (see page 65)

DeleteRange (see page 69)

Extract (see page 73)

InsertRange (see page 79)

Remove (see page 87)

OnNotify (see page 85)

34
Generics.Collections.TList.InsertRange

Insert collection into list.

Description

```
Pascal
            procedure InsertRange(Index: Integer; const Values: array of T); overload;
            procedure InsertRange(Index: Integer; const Collection: IEnumerable<T>); overload;
            procedure InsertRange(Index: Integer; const Collection: TEnumerable<T>); overload;
C++
          __fastcall InsertRange(int Index, T const *Values, const int Values_Size)/* overload
*/;
          __fastcall InsertRange(int Index, const
System::DelphiInterface<System::IEnumerable__1<T> >  Collection)/* overload */;
          __fastcall InsertRange(int Index, const TEnumerable__1<T>* Collection)/* overload */;
```

InsertRange inserts an array of values **Values** into the list at the index **Index**. If the list **Count** plus the extra entries is greater than **Capacity**, **Capacity** is increased.

If **Index** is not valid for the list, an **EArgumentOutOfRangeException** exception is raised.

This is an O(n + m) operation, where n is the number of items in the list and m is the number of entries in **Values**.

An OnNotify event occurs indicating items were inserted in the list.

See Also

Count (see page 59)

Capacity (see page 53)

Delete (see page 65)

DeleteRange (see page 69)

Extract (see page 73)

InsertRange

Remove (see page 87)

OnNotify (see page 85)

35 Generics.Collections.TList.Items

Item at index.

Description

Pascal

```
property Items[Index: Integer]: T;
```

C++

```
__property T Items[int Index];
```

Items gets or sets the list element at the specified index.

You can use Items to get and set list values by index using the syntax `myList[i]` to access the ith item of the list.

See Also

Capacity (see page 53)

Count (see page 59)

36
Generics.Collections.TList.LastIndexOf

Get index of last instance of entry.

Description

Pascal

```
function LastIndexOf(const Value: T): Integer;
```

C++

```
int __fastcall LastIndexOf(const T Value);
```

LastIndexOf gets the zero-based index of the last instance of **Value** in the list. If not found, the function returns -1.

Since the search is linear, it is an O(n) operation for a list with n entries.

See Also

IndexOf (see page 75)

37 Generics.Collections.TList.OnNotify

Occurs when list changes.

Description

```
Pascal
          property OnNotify: TCollectionNotifyEvent<T>;
C++
        __property _decl_TCollectionNotifyEvent__1(T, OnNotify);
```

The OnNotify event occurs when items are added or removed from the list. Multiple events may occur for a single operation. This allows removed objects to be freed.

See Also

Add (see page 45)

AddRange (see page 49)

Delete (see page 65)

DeleteRange (see page 69)

Insert (see page 77)

InsertRange (see page 79)

Remove (see page 87)

OnNotify (see page 139)

OnNotify (see page 157)

TCollectionNotifyEvent (see page 3)

TCollectionNotification (see page 1)

38 Generics.Collections.TList.Remove

Remove first occurrence of value.

Description

Pascal

```
function Remove(const Value: T): Integer;
```

C++

```
int __fastcall Remove(const T Value);
```

Remove removes the first instance of **Value** in the list, returning its zero-based index. If **Value** is not in the list, this function returns -1.

Since the search is linear, it is an O(n) operation for a list with n entries.

An OnNotify event occurs indicating an entry was removed from the list.

See Also

Clear (see page 55)

Delete (see page 65)

DeleteRange (see page 69)

Extract (see page 73)

IndexOf (see page 75)

OnNotify (see page 85)

39 Generics.Collections.TList.Reverse

Reverse list order.

Description

Pascal

```
procedure Reverse;
```

C++

```
__fastcall Reverse();
```

This method reverses the order of all list elements.

It is a O(n) operation, where n is the number of list elements.

See Also

Sort (see page 91)

40 Generics.Collections.TList.Sort

Sort list.

Description

```
Pascal
            procedure Sort; overload;
            procedure Sort(const AComparer: IComparer<T>); overload;
C++
          __fastcall Sort(void)/* overload */;
          __fastcall Sort(const System::DelphiInterface<Generics_defaults::IComparer__1<T> >
AComparer)/* overload */;
```

This method sorts the list. If **AComparer** is provided, it is used to compare elements; otherwise the default comparator for the list elements is used.

This sort is a O(n log n) operation, where n is the number of list elements. A QuickSort algorithm is used, so the order of equal elements may not be preserved.

Delphi Examples:

```
{
This example requires a button and two TListboxes on a form.  The
Generics.Collections objects are created dynamically.  Notice that you can
call the TList Soft method with the TComparer as a parameter, or create the
TList with the TComparer and just call Sort.  Also, do not free reverseComp
after associating it with sortlist.
}
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Generics.Defaults,
    Generics.Collections;   // must appear after Classes to use the correct
TCollectionNotification

type
    TForm2 = class(TForm)
        Button1: TButton;
        ListBox1: TListBox;
        ListBox2: TListBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        ListBox3: TListBox;
```

```
        ListBox4: TListBox;
        Label4: TLabel;
        ListBox5: TListBox;
        Label5: TLabel;
        ListBox6: TListBox;
        Label6: TLabel;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
        Dictionary: TDictionary<Integer, Integer>;
        procedure OnDictAdd(
            Sender: TObject; const Item: Integer; Action: TCollectionNotification);
    end;
    TReverseIntComparer = class(TComparer<Integer>)
        function Compare(const Left, Right: Integer): Integer; override;
    end;

var
    Form2: TForm2;

implementation

{$R *.dfm}
{$APPTYPE CONSOLE}

function TReverseIntComparer.Compare(const Left, Right: Integer): Integer;
begin
    if Left < Right then
        Result := 1
    else if Left > Right then
        Result := -1
    else
        Result := 0;
end;

procedure TForm2.OnDictAdd(Sender: TObject; const Item: Integer; Action:
TCollectionNotification);
begin
    if Action = cnAdded then
        WriteLn('TDictionary Key has been added to the dictionary');
    if Action = cnRemoved then
        WriteLn('TDictionary Key has been removed from the dictionary');
end;

procedure TForm2.Button1Click(Sender: TObject);
var
    mylist, sortlist: TList<Integer>;
    reverseComp: TReverseIntComparer;
    myarray: TArray;
    values: array of Integer;
    i: Integer;
begin
    myarray:= TArray.Create;
    Dictionary.Add(0, 0);
    try
        Randomize;
        mylist:= Generics.Collections.TList<Integer>.Create;
        reverseComp:= TReverseIntComparer.Create;
        for i := 0 to 100 - 1 do
            mylist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox1.Items.Add(IntToStr(mylist[i]));
        mylist.Sort;
```

```
        for i := 0 to mylist.Count - 1 do
            Listbox2.Items.Add(IntToStr(mylist[i]));
        mylist.Delete(0);
        mylist.Delete(2);
        mylist.Delete(4);
        for i := 0 to mylist.Count - 1 do
            Listbox3.Items.Add(IntToStr(mylist[i]));
        mylist.Sort(reverseComp);
        for i := 0 to mylist.Count - 1 do
            Listbox4.Items.Add(IntToStr(mylist[i]));
        sortlist:= Generics.Collections.TList<Integer>.Create(reverseComp);
        for i := 0 to 100 - 1 do
            sortlist.Add(Random(100));
        for i := 0 to mylist.Count - 1 do
            Listbox5.Items.Add(IntToStr(sortlist[i]));
        sortlist.Sort;
        for i := 0 to mylist.Count - 1 do
            Listbox6.Items.Add(IntToStr(mylist[i]));
    finally
//    reverseComp.Free;
        sortlist.Free;
        mylist.Free;
        Dictionary.Free;
        myarray.Free;
    end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
    Dictionary := TDictionary<Integer, Integer>.Create;
    Dictionary.OnKeyNotify := Form2.OnDictAdd;
end;
```

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    // Prints type info for string type
    TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
    GString->PrintTypeInfo(Memo1);

    // Prints type info for Byte type
    TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
    GByte->PrintTypeInfo(Memo1);

    // Prints type info for Double type
    TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
    GDouble->PrintTypeInfo(Memo1);

    // Prints type info for "array of String" type
    // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of
string>();
    // GStringArray->PrintTypeInfo(Memo1);
}
```

See Also

BinarySearch (see page 51)

Reverse (see page 89)

41 Generics.Collections.TList.TrimExcess

Set list capacity to number of list elements.

Description

```
Pascal
            procedure TrimExcess;
C++
          __fastcall TrimExcess();
```

TrimExcess sets Capacity to Count, getting rid of all excess capacity in the list.

See Also

Count (see page 59)

Capacity (see page 53)

Clear (see page 55)

TrimExcess (see page 35)

TrimExcess (see page 143)

TrimExcess (see page 165)

42
Generics.Collections.TObjectDictionary

Collection of key-value pairs of objects.

Description

TObjectDictionary represents a generic collection of key-value pairs of objects.

TObjectDictionary is a TDictionary with the capability of automatically freeing objects when they are removed from the dictionary. When a TObjectDictionary instantiated, an **Ownerships** parameter specifies whether the dictionary owns the keys and/or values. If the key and/or value is owned by the dictionary, when the entry is removed the key and/or value object is freed.

See Also

TDictionary (see page 5)

Create (see page 99)

43
Generics.Collections.TObjectDictionary.Create

Create TObjectDictionary instance.

Description

```
Pascal
            constructor Create(Ownerships: TDictionaryOwnerships; ACapacity: Integer = 0);
overload;
            constructor Create(Ownerships: TDictionaryOwnerships;  const AComparer:
IEqualityComparer<TKey>); overload;
            constructor Create(Ownerships: TDictionaryOwnerships; ACapacity: Integer;  const
AComparer: IEqualityComparer<TKey>); overload;
C++
          __fastcall TObjectDictionary__2(TDictionaryOwnerships Ownerships, int ACapacity)/*
overload */;
          __fastcall TObjectDictionary__2(TDictionaryOwnerships Ownerships,  const
System::DelphiInterface<Generics_defaults::IEqualityComparer__1<TKey> >  AComparer)/* overload
*/;
          __fastcall TObjectDictionary__2(TDictionaryOwnerships Ownerships, int ACapacity,
const System::DelphiInterface<Generics_defaults::IEqualityComparer__1<TKey> >  AComparer)/*
overload */;
```

This overloaded method creates a TObjectDictionary instance.

The **Ownerships** parameter is a TDictionaryOwnerships that indicates whether the keys and/or values in entries are owned by the dictionary. Either the key or value or both or neither may be owned by the dictionary. If the object is owned, when the entry is removed from the dictionary, the key and/or value is freed.

ACapacity is the initial capacity of the dictionary.

AComparer is an equality comparator function. If not provided, the default comparator for the type is used.

See Also

TDictionaryOwnerships (see page 39)

Create (see page 21)

44 Generics.Collections.TObjectList

Ordered list of objects.

Description

TObjectList represents an ordered list of objects, accessible by an index.

TObjectList is a TList with the capability of automatically freeing object entries when they are removed from the list. When a TObjectList instantiated, an **AOwnsObjects** parameter specifies whether the list owns the list entries. If the entry is owned, when the entry object is removed from the list, the entry object is freed.

The OwnsObjects property gets or sets the object ownership.

See Also

OwnsObjects (see page 105)

Create (see page 103)

45
Generics.Collections.TObjectList.Create

Create TObjectList instance.

Description

```
Pascal
            constructor Create(AOwnsObjects: Boolean = True); overload;
            constructor Create(const AComparer: IComparer<T>; AOwnsObjects: Boolean = True);
overload;
            constructor Create(Collection: TEnumerable<T>; AOwnsObjects: Boolean = True);
overload;
C++
          __fastcall TObjectList__1(bool AOwnsObjects)/* overload */;
          __fastcall TObjectList__1(const
System::DelphiInterface<Generics_defaults::IComparer__1<T> >  AComparer, bool AOwnsObjects)/*
overload */;
          __fastcall TObjectList__1(TEnumerable__1<T>* Collection, bool AOwnsObjects)/*
overload */;
```

This overloaded method creates a TObjectList instance.

The **AOwnsObjects** parameter is a boolean that indicates whether object entries are owned by the list. If the object is owned, when the entry is removed from the list, the object is freed. The OwnsObjects property is set from the value of this parameter. The default is true.

Collection is a collection with which to initialize the list. The objects are added in the same order as in **Collection**. If **Collection** is specified, the creation is an O(n) operation, where n is the number of items in **Collection**.

AComparer is an equality comparator function. If not provided, the default comparator for the type is used.

See Also

OwnsObjects (see page 105)

Create (see page 61)

46
Generics.Collections.TObjectList.OwnsObjects

Get or set object ownership.

Description

```
Pascal
          property OwnsObjects: Boolean;
C++
        __property bool OwnsObjects;
```

OwnsObjects gets or sets whether objects in the list are owned by the list or not. If entries are owned, when an entry object is removed from the list, the entry object is freed. Create initializes this property.

See Also

Create (see page 103)

47 Generics.Collections.TObjectQueue

Queue of objects.

Description

TObjectQueue represents a queue of objects.

TObjectQueue is a TQueue with the capability of automatically freeing object entries when they are removed from the queue. When a TObjectQueue instantiated, an **AOwnsObjects** parameter specifies whether the queue owns the queue entries. If the entry is owned, when the entry object is removed from the queue, the entry object is freed.

The OwnsObjects property gets or sets the object ownership.

See Also

TQueue (see page 123)

OwnsObjects (see page 113)

Create (see page 109)

Dequeue (see page 111)

48 Generics.Collections.TObjectQueue.Create

Create TObjectQueue instance.

Description

```
Pascal
            constructor Create(AOwnsObjects: Boolean = True); overload;
            constructor Create(Collection: TEnumerable<T>; AOwnsObjects: Boolean = True);
overload;
C++
          __fastcall TObjectQueue__1(bool AOwnsObjects)/* overload */;
          __fastcall TObjectQueue__1(TEnumerable__1<T>* Collection, bool AOwnsObjects)/*
overload */;
```

This overloaded method creates a TObjectQueue instance.

The **AOwnsObjects** parameter is a boolean that indicates whether object entries are owned by the queue. If the object is owned, when the entry is removed from the queue, the object is freed. The OwnsObjects property is set from the value of this parameter. The default is true.

Collection is a collection with which to initialize the queue. The objects are added in the same order as in **Collection**. If **Collection** is specified, the creation is an O(n) operation, where n is the number of items in **Collection**.

See Also

OwnsObjects (see page 113)

Create (see page 129)

49 Generics.Collections.TObjectQueue.Dequeue

Remove top queue item.

Description

```
Pascal
          procedure Dequeue;
C++
         __fastcall Dequeue();
```

Dequeue removes but does *not* return the top element of the queue. Count is decremented by 1. If Count is already 0, an error is raised.

Note: TObjectQueue.Dequeue differs from TQueue.Dequeue in that it is a procedure and does not return the dequeued element. Otherwise, both methods function similarly. Use TQueue.Peek to work with the head of the queue and TQueue.Dequeue when finished with the head, or alternatively use TQueue.Extract to take ownership.

See Also

Dequeue (see page 131)

Extract (see page 137)

Peek (see page 141)

50
Generics.Collections.TObjectQueue.OwnsObjects

Get or set object ownership.

Description

```
Pascal
          property OwnsObjects: Boolean;
C++
        __property bool OwnsObjects;
```

OwnsObjects gets or sets whether objects in the queue are owned by the queue or not. If entries are owned, when an entry object is removed from the queue, the entry object is freed. Create initializes this property.

See Also

Create (see page 109)

51 Generics.Collections.TObjectStack

Last in, first out stack of objects.

Description

TObjectStack represents a last in, first out stack of objects of the same type. It is of arbitrary size, expanding as needed. You can push **nil** on the stack.

TObjectStack is a TStack with the capability of automatically freeing object entries when they are removed from the stack. When a TObjectStack instantiated, an **AOwnsObjects** parameter specifies whether the stack owns the stack entries. If the entry is owned, when the entry object is removed from the stack, the entry object is freed.

The OwnsObjects property gets or sets object ownership.

See Also

TStack (see page 145)

OwnsObjects (see page 119)

Create (see page 117)

Pop (see page 121)

52
Generics.Collections.TObjectStack.Create

Create TObjectStack instance.

Description

```
Pascal
            constructor Create(AOwnsObjects: Boolean = True); overload;
            constructor Create(Collection: TEnumerable<T>; AOwnsObjects: Boolean = True);
overload;
C++
          __fastcall TObjectStack__1(bool AOwnsObjects)/* overload */;
          __fastcall TObjectStack__1(TEnumerable__1<T>* Collection, bool AOwnsObjects)/*
overload */;
```

This overloaded method creates a TObjectStack instance.

The **AOwnsObjects** parameter is a boolean that indicates whether object entries are owned by the stack. If the object is owned, when the entry is removed from the stack, the object is freed. The OwnsObjects property is set from the value of this parameter. The default is true.

Collection is a collection with which to initialize the stack. The objects are pushed on the stack in the same order as in **Collection**. If **Collection** is specified, the creation is an O(n) operation, where n is the number of items in **Collection**.

See Also

OwnsObjects (see page 119)

Create (see page 151)

53 Generics.Collections.TObjectStack.OwnsObjects

Get or set object ownership.

Description

```
Pascal
          property OwnsObjects: Boolean
C++
        __property bool OwnsObjects
```

OwnsObjects gets or sets whether objects in the stack are owned by the stack or not. If entries are owned, when an entry object is removed from the stack, the entry object is freed. Create initializes this property.

See Also

Create (see page 117)

54 Generics.Collections.TObjectStack.Pop

Pop stack item.

Description

```
Pascal
            procedure Pop;
C++
          __fastcall Pop();
```

This method removes one item from the top of the stack *without* returning it. Count is decremented by 1. If Count is already 0, an error is raised.

Note: TObjectStack.Pop differs from TStack.Pop in that it is a procedure and does not return the popped element. Otherwise, both methods function similarly. Use TStack.Peek to work with the head of the stack and TStack.Pop when finished with the head, or alternatively use TStack.Extract to take ownership.

See Also

Extract (see page 155)

Peek (see page 159)

Pop (see page 161)

55 Generics.Collections.TQueue

Queue implemented over array, using wrapping.

Description

TQueue is a queue implemented over array, using wrapping.

You can add items to the end of the queue and remove them from the start or remove all the items. You can also peek at the top item without removing it. You can add **nil** objects to the queue.

Count contains the number of items in the queue.

An OnNotify event tells you when the queue has changed.

The class TObjectQueue inherits from TQueue and provides an automatic mechanism for freeing objects removed from queues.

See Also

TObjectQueue (see page 107)

Count (see page 127)

Clear (see page 125)

Dequeue (see page 131)

Enqueue (see page 135)

Extract (see page 73)

Peek (see page 141)

OnNotify (see page 139)

56 Generics.Collections.TQueue.Clear

Empty queue.

Description

```
Pascal
            property Count: Integer;
C++
          __property int Count;
```

Clear removes all entries from the queue. Count is set to 0. This does not change the capacity. This is a O(n) operation where n is the length of the queue.

Note: Clear does not free the items as they are dequeued. If you need to free them, use the OnNotify event, which occurs for every item dequeued and provides the dequeued item.

See Also

Count (see page 127)

Dequeue (see page 131)

Destroy (see page 133)

OnNotify (see page 139)

57 Generics.Collections.TQueue.Count

Number of queue elements.

Description

Count gets the number of elements in the queue. This property cannot be set.

58 Generics.Collections.TQueue.Create

Create queue.

Description

```
Pascal
            constructor Create(Collection: TEnumerable<T>);
C++
          __fastcall TQueue__1(TEnumerable__1<T>* Collection);
```

This method creates and initializes a TQueue instance. Each item in collection **Collection** is added to the end of the queue (enqueued) in the same order as in **Collection**.

See Also

Destroy (see page 133)

Enqueue (see page 135)

59 Generics.Collections.TQueue.Dequeue

Remove top queue item.

Description

```
Pascal
            function Dequeue: T;
C++
          T __fastcall Dequeue();
```

Dequeue removes and returns the top element of the queue. Count is decremented by 1. If Count is already 0, an error is raised.

An OnNotify event occurs indicating an item was removed from the queue. Dequeue is the same as Extract except for the event code indicating an element was removed rather than extracted.

Dequeue functions similarly to Peek except that Dequeue removes an element from the queue.

This is a O(1) operation.

See Also

Count (see page 127)

Enqueue (see page 135)

Extract (see page 137)

Peek (see page 141)

OnNotify (see page 139)

60 Generics.Collections.TQueue.Destroy

Destroy queue.

Description

```
Pascal
            destructor Destroy; override;
C++
          __fastcall virtual ~TQueue__1();
```

This method clears the queue using Clear and destroys it.

Note: Clear does not free the items as they are dequeued. If you need to free them, use the OnNotify event, which occurs for every item dequeued and provides the dequeued item.

See Also

Clear (see page 125)

Create (see page 129)

61 Generics.Collections.TQueue.Enqueue

Add item to end of queue.

Description

```
Pascal
            procedure Enqueue(const Value: T);
C++
          __fastcall Enqueue(const T Value);
```

Enqueue adds the given item **Value** to the end of the queue. You can enqueue **nil**. Count is incremented by 1. If Count is at capacity, the queue size is automatically increased. Count is incremented by 1.

An OnNotify event occurs indicating an item was added to the queue.

This is a O(1) operation, unless the capacity must increase. In this case, it is O(n), where n is Count.

See Also

Count (see page 127)

Dequeue (see page 131)

OnNotify (see page 139)

62 Generics.Collections.TQueue.Extract

Remove top queue item.

Description

```
Pascal
            function Extract: T;
C++
          T __fastcall Extract();
```

Extract removes and returns the top element of the queue. Count is decremented by 1. If Count is already 0, an error is raised.

An OnNotify event occurs indicating an item was removed from the queue. Extract is the same as Dequeue except for the event code indicating an element was extracted rather than removed.

Extract functions similarly to Peek except that Extract removes an element from the queue.

Extract is similar to Dequeue and is provided so items may be removed without freeing.

This is a O(1) operation.

See Also

Count (see page 127)

Dequeue (see page 131)

Enqueue (see page 135)

Peek (see page 141)

OnNotify (see page 139)

63 Generics.Collections.TQueue.OnNotify

Occurs when queue changes.

Description

```
Pascal
            property OnNotify: TCollectionNotifyEvent<T>;
C++
          __property _decl_TCollectionNotifyEvent__1(T, OnNotify);
```

The OnNotify event occurs when items are added or removed from the queue. This allows removed objects to be freed.

See Also

Dequeue (see page 131)

Enqueue (see page 135)

Extract (see page 137)

OnNotify (see page 85)

OnNotify (see page 157)

TCollectionNotifyEvent (see page 3)

TCollectionNotification (see page 1)

64 Generics.Collections.TQueue.Peek

Get top item on queue.

Description

```
Pascal
            function Peek: T;
C++
          T __fastcall Peek();
```

Peek returns the top item of the queue without removing it. Count is unchanged. No event is generated. If Count is 0, an error is raised.

Peek functions similarly to Dequeue except that Dequeue removes the top item.

This is a O(1) operation.

See Also

Count (see page 127)

Dequeue (see page 131)

Enqueue (see page 135)

Extract (see page 137)

65
Generics.Collections.TQueue.TrimExcess

Set capacity to number of queue elements.

Description

```
Pascal
            procedure TrimExcess;
C++
          __fastcall TrimExcess();
```

TrimExcess sets the queue capacity to Count.

See Also

Count (see page 127)

TrimExcess (see page 35)

TrimExcess (see page 95)

TrimExcess (see page 165)

66 Generics.Collections.TStack

Last in, first out stack.

Description

TStack represents a last in, first out stack of the same type. It is of arbitrary size, expanding as needed. You can push **nil** on the stack.

When the stack changes, an OnNotify event is generated.

Count contains the number of stack entries.

The class TObjectStack inherits from TStack and provides an automatic mechanism for freeing objects removed from stacks.

See Also

TObjectStack (see page 115)

Count (see page 149)

Clear (see page 147)

Pop (see page 161)

Push (see page 163)

TrimExcess (see page 165)

OnNotify (see page 157)

67 Generics.Collections.TStack.Clear

Clear stack.

Description

```
Pascal
            procedure Clear;
C++
          __fastcall Clear();
```

Clear pops all entries from the stack. Count, the number of entries on the stack, is set to zero. The capacity is also set to zero. This is an O(n) operation where n is Count.

Note: Clear does not free the items as they are popped. If you need to free them, use the OnNotify event, which occurs for every pop and provides the popped item.

See Also

Count (see page 149)

Destroy (see page 153)

Pop (see page 161)

OnNotify (see page 157)

68 Generics.Collections.TStack.Count

Number of stack entries.

Description

```
Pascal
            property Count: Integer;
C++
          __property int Count;
```

Count gets the number of entries on the stack. You cannot write Count.

See Also

Clear (see page 147)

Pop (see page 161)

Push (see page 163)

TrimExcess (see page 165)

69 Generics.Collections.TStack.Create

Create stack.

Description

```
Pascal
            constructor Create(Collection: TEnumerable<T>); overload;
C++
          __fastcall TStack__1(TEnumerable__1<T>* Collection);
```

This method creates and initializes a TStack instance.

Collection is a collection whose objects are pushed onto the stack in the order they are in **Collection**.

Create is an O(n) operation, where n is the number of elements in **Collection**.

See Also

Destroy (see page 153)

70 Generics.Collections.TStack.Destroy

Destroy stack instance.

Description

```
Pascal
            destructor Destroy; override;
C++
          __fastcall virtual ~TStack__1();
```

This method clears a stack using Clear and destroys it.

Note: Clear does not free the items as they are popped. If you need to free them, use the OnNotify event, which occurs for every pop and provides the popped item.

See Also

Clear (see page 147)

Create (see page 151)

71 Generics.Collections.TStack.Extract

Remove top stack item.

Description

```
Pascal
            function Extract: T;
C++
          T __fastcall Extract();
```

Extract removes and returns the top element of the stack. Count is decremented by 1. If Count is already 0, an error is raised.

An OnNotify event occurs indicating an item was removed from the stack. Extract is the same as Pop except for the event code indicating an element was extracted rather than removed.

Extract functions similarly to Peek except that Extract removes an element from the stack.

Extract is similar to Pop and is provided so items may be removed without freeing.

This is a O(1) operation.

See Also

Count (see page 149)

Pop (see page 161)

Push (see page 163)

OnNotify (see page 157)

72 Generics.Collections.TStack.OnNotify

Occurs when stack changes.

Description

Pascal

```
property OnNotify: TCollectionNotifyEvent<T>;
```

C++

```
__property _decl_TCollectionNotifyEvent__1(T, OnNotify);
```

The OnNotify event occurs when items are added to or removed from the stack. This allows removed objects to be freed.

See Also

OnNotify (see page 85)

OnNotify (see page 139)

TCollectionNotifyEvent (see page 3)

TCollectionNotification (see page 1)

73 Generics.Collections.TStack.Peek

Look at top stack item.

Description

```
Pascal
            function Peek: T;
C++
          T __fastcall Peek();
```

Peek returns the item from the top of the stack without removing it. If Count is 0, an error is raised. Count is unchanged.

Pop functions similarly to Peek except that Pop removes an element from the stack.

This is a O(1) operation.

See Also

Count (see page 149)

Pop (see page 161)

74 Generics.Collections.TStack.Pop

Pop stack item.

Description

```
Pascal
            function Pop: T;
C++
          T __fastcall Pop();
```

This method removes one item from the top of the stack and returns it. Count is decremented by 1. If Count is already 0, an error is raised.

An OnNotify event occurs indicating an item was removed from the stack. Pop is the same as Extract except for the event code indicating an element was removed rather than extracted.

Pop functions similarly to Peek except that Pop removes an element from the stack.

This is a O(1) operation.

See Also

Count (see page 149)

Clear (see page 147)

Extract (see page 155)

Push (see page 163)

OnNotify (see page 157)

75 Generics.Collections.TStack.Push

Push stack item.

Description

Pascal

```
procedure Push(const Value: T);
```

C++

```
__fastcall Push(const T Value);
```

This method inserts one item onto the top of the stack. Count is incremented by 1. You can push **nil** onto the stack. The stack's capacity is automatically increased if necessary.

An OnNotify event occurs indicating an item was added to the stack.

This is a O(1) operation unless the capacity must increase. In this case, it is O(n), where n is Count.

See Also

Count (see page 149)

Extract (see page 155)

Pop (see page 161)

OnNotify (see page 157)

76
Generics.Collections.TStack.TrimExcess

Set capacity same as current number of items.

Description

```
Pascal
            procedure TrimExcess;
C++
          __fastcall TrimExcess();
```

TrimExcess sets the capacity equal to the number of items on the stack.

See Also

Count (see page 149)

TrimExcess (see page 35)

TrimExcess (see page 95)

TrimExcess (see page 143)

77 Generics.Collections

Generics.Collections implements the new generics feature introduced in Delphi 2009. C++Builder handles Delphi generics as templates, and the generic types must be instantiated on the Delphi side.

Index

G