Consultingwerk
software architecture and development

# What are user-defined generic types and why should I use them?

PUG Challenge 2026 Amsterdam

Peter Judge, Consultingwerk
peter.judge@consultingwerk.com

# Peter Judge




Writing 4GL since 1996, working on a variety of frameworks and applications

Focusing on integration with OpenEdge applications, PASOE, software architecture and web technologies

Active member of the OpenEdge community and speaker at international conferences

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Consultingwerk
software architecture and development

# Our Focus: Modernizing OpenEdge Applications



**Modernization of Legacy OpenEdge Applications**



**Deep Technical Expertise**



**Global IT Partner with Local Presence**




**More than Consulting – We Deliver Tools & Solutions**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Previously, on PUG Challenge …

2022: “Generic <T>ype collections in ABL”, slide 16

~~As of 12.5, you cannot define your own generic types / collections~~




**In 2022 my AnimalList<T as Animal> was pseudo-code. In 13.0 it is the real syntax.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Agenda

| | | |
|---|---|---|
| 1 | **Why?** | The same pattern, built again and again |
| 2 | **How?** | Classes, interfaces, constraints |
| 3 | **The rules** | What you can and can't do with T |
| 4 | **Under the hood** | Compiler, runtime, r-code, interop |
| 5 | **In the real world** | SmartComponent Library lists and dictionaries, and friends |
| 6 | **Gotchas** | And when generics are the wrong tool |

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

Consultingwerk
software architecture and development

# Built the Same Pattern More Than Once? We Did.

**131**

list classes
(109 direct subclasses of `GenericList`)

**36**

dictionaries
(16 direct subclasses of `GenericDictionary`)

**1**

include file per class:
same code, different type

### Generate one class per type

Include files, scaffolding, copy-paste
Typed, but the same code 167 times

### Or one class for Progress.Lang.Object

Written once, but `TYPE-OF` at runtime
and a `CAST` in every caller

**Both work. Neither lets the compiler help you. That is what generics fix.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

The full stack modernization framework
SmartComponent Library

# Speaking Generic

```
class AnimalList<T as Animal>:                          /* the generic type  */
var AnimalList<Cat> oCats = new AnimalList<Cat>(). /* a constructed type */
```

| Term | In the code | What it is |
|---|---|---|
| **Generic type** | `AnimalList<T as Animal>` | A class or interface with type parameters |
| **Type parameter** | `T` | Placeholder, used like a type inside the class |
| **Type constraint** | `as Animal` | What T may be: Animal or a subtype |
| **Type argument** | `Cat` | The type the client supplies |
| **Constructed type** | `AnimalList<Cat>` | “AnimalList of Cat”: what you define and NEW |

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Agenda

| | | |
|---|---|---|
| 1 | **Why?** | The same pattern, built again and again |
| 2 | **How?** | Classes, interfaces, constraints |
| 3 | **The rules** | What you can and can't do with T |
| 4 | **Under the hood** | Compiler, runtime, r-code, interop |
| 5 | **In the real world** | SmartComponent Library lists and dictionaries, and friends |
| 6 | **Gotchas** | And when generics are the wrong tool |

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Your First Generic Class

13.0+

**AnimalList.cls**

```
using Demo.Animals.Animal.
using Progress.Collections.List.

class Demo.Collections.Generic.AnimalList<T as Animal>:
  define private variable mItems as List<T> no-undo.

  constructor public AnimalList():
    mItems = new List<T>().
  end constructor.

  method public void Add(input pAnimal as T):
    mItems:Add(pAnimal).
  end method.

  method public T Get(input pIndex as integer):
    return mItems:Get(pIndex).
  end method.
end class.
```

## What's new here

- `<T as Animal>` after the class name
- T passed on: `List<T>`
- `Get()` returns a T
- Inside, T is an Animal: `Name`, `Sound()` work
- One .cls file, one .r file

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Generic Interfaces

IRepository.cls

```
interface IRepository<T as IEntity>:
  define public property Count
    as integer no-undo get.
  method public void Save(
    input pEntity as T).
  method public Result<T> FindById(
    input pId as character).
end interface.
```

… and the classes and interfaces that use it

```
class ListRepository<T as IEntity>
  implements IRepository<T>:

/* inherit with the same T */
interface IManager<T as Employee>
  inherits IEmployee<T>:

/* built-in generic interface */
class Person
  implements IComparable<Person>:
```

**Interfaces take type parameters too, and pass them on**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# What Can T Be?

## Allowed as the constraint

An ABL class: `T as Animal`

An interface: `T as IEntity`

A constructed generic type, fully qualified:

```
T as Progress.Collections.
  List<Demo.Animals.Animal>
```

Several parameters:

```
<K as IHashable, V as …>
```

Exactly one constraint each

## Not allowed

Primitives: `<T as character>`

.NET types: `<T as System.Object>`

Handle-based objects

No constraint at all: write

```
T as Progress.Lang.Object
```

Recursive: `T as IComparable<T>`

Type arguments follow the same rules

**If it isn't an ABL object, it can't be a T**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Mix and Match

```
/* non-generic class, generic interface */
class Person implements IComparable<Person>:

/* generic class, non-generic base class */
class RepositoryOnBase<T as IEntity> inherits RepositoryBase:

/* generic class, generic base class */
class GenericSub<T as Animal> inherits AnimalList<T>:

/* FINAL, SERIALIZABLE: yes. ABSTRACT: no (error 20978) */
```

**Non-generic class inherits AnimalList<Cat>? Compiles, but can't be used in 13.0.0**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Live Demo 1: Your First Generic Class





## Before

- One list for any object
- `CAST` in every caller
- Gold fails at runtime

» **Code:**

```
var AnimalList<Cat> oCats
  = new AnimalList<Cat>().
oCats:Add(new Cat("Muschel")).
oCats:Add(new Cat("Puschel")).
oCat = oCats:Get(2).

/* oCats:Add(new Dog("Rex")). */
```



## You'll see

- No `CAST`
- Dog and Gold refused by the compiler

**The error moved from production to your compiler**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

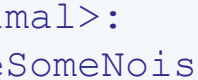
# Agenda

| | | |
|---|---|---|
| 1 | **Why?** | The same pattern, built again and again |
| 2 | **How?** | Classes, interfaces, constraints |
| 3 | **The rules** | What you can and can't do with T |
| 4 | **Under the hood** | Compiler, runtime, r-code, interop |
| 5 | **In the real world** | SmartComponent Library lists and dictionaries, and friends |
| 6 | **Gotchas** | And when generics are the wrong tool |

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Inside the Class, T Is Its Constraint

```
class AnimalList<T as Animal>:
  method public void MakeSomeNoise():
    define variable i as integer no-undo.
    do i = 1 to mItems:Count:
      message mItems:Get(i):Name
              mItems:Get(i):Sound().
    end.
  end method.
```



## With a T you can

- Use the constraint's members
- Assign it to a T, or to a variable of the constraint type, and back: the compiler lets an Animal into a T
- Compare: `=` (same object) or `Equals()`
- Pass T on: `new List<T>()`, `new Result<T>(…)`

**Inside the class, T is its constraint, both ways: return new Dog() compiles in AnimalList<Cat>'s code**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# NEW T()? Not So Fast

**What you want**

```
class Factory<T as Animal>:
  method public T Make():
    return new T().
    /* compile error */
  end method.
end class.
```

**What works: reflection**

```
constructor public Factory():
  var Progress.Lang.Class[] oArgs.
  oArgs = this-object:GetClass()
            :GetTypeArguments().
  mType = oArgs[1].
end constructor.
method public T Create():
  return dynamic-cast(mType:New(),
                      mType:TypeName).
```

**new List<T>() is fine. new T() isn't: reflection works, but checks at runtime.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Quiz: Which Admit() Wins?

```
class Shelter<T as Animal>:
  method public char Admit(input p as T):
    return "#1 (T)".
  end method.
  method public char Admit(input p as Cat):
    return "#2 (Cat)".
  end method.
end class.
```

```
var Shelter<Dog> oDogs
  = new Shelter<Dog>().
oDogs:Admit(new Dog()).

var Shelter<Cat> oCats
  = new Shelter<Cat>().
oCats:Admit(new Cat()).
```

**<Dog>: #1, only Admit(T) matches.  <Cat>: compile error 13843, ambiguous call.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Quiz: Is a List<Cat> a List<Animal>?

```
var IList<Animal> oAnimals = new List<Cat>().    /* compiles? */
var IList<Cat>    oCats    = new List<Cat>().    /* fine */
```



**No. Type arguments must match exactly.**

Otherwise `oAnimals:Add(new Dog())` would put a dog among the cats.

Same for `CAST` and `DYNAMIC-CAST`.

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# TYPE-OF, CAST, GET-CLASS: Use the Constraint

**Not allowed**

```
/* inside class X<T as Animal> */
if type-of(o, T) then
  cast(o, T):Sound().

oCls = get-class(T).

/* all three: compile errors */
```

**Allowed**

```
/* use the constraint type */
if type-of(o, Animal) then
  cast(o, Animal):Sound().

/* or ask the runtime */
oArgs = this-object:GetClass()
          :GetTypeArguments().
```

**T by itself is not a type you can test or cast to. Its constraint is.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# The Small Print

### No STATIC members

Put them in a non-generic base class

### CATCH on T: compiles

Docs say no; it catches by the constraint

Generic error type: no (20518)

### No generic ENUMs

ENUM types cannot be generic

### No .NET types

Not as constraint, not as type argument

### SERIALIZABLE

A member of type T needs a serializable constraint (17535)

Type argument: checked at runtime (17586)

### No ABSTRACT

A generic class can't be abstract (20978)

**No statics, no generic methods, no abstract: no static Result:Ok<T>() factory**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Agenda

| | | |
|---|---|---|
| 1 | **Why?** | The same pattern, built again and again |
| 2 | **How?** | Classes, interfaces, constraints |
| 3 | **The rules** | What you can and can't do with T |
| 4 | **Under the hood** | Compiler, runtime, r-code, interop |
| 5 | **In the real world** | SmartComponent Library lists and dictionaries, and friends |
| 6 | **Gotchas** | And when generics are the wrong tool |

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# The Compiler Says No

| Code | Error on 13.0.0 |
|---|---|
| `AnimalList<Cat>:Add(new Dog())` | 12905 Parameter 1 for METHOD Add is not type compatible |
| `AnimalList<Gold>` | 20504 argument 1 (Gold) must inherit from Animal |
| `AnimalList<character>` | 12886 Could not find class or interface character |
| `IList<Animal> = new List<Cat>()` | 223 Incompatible data types |
| `Shelter<Cat>:Admit(cat)` | 13843 Ambiguous method call |
| `new T(),type-of(x, T)` | 20524 Type Parameter T may not be used by itself |
| STATIC member | 20975 A generic class cannot have STATIC members |
| ABSTRACT generic class | 20978 Generic class cannot also be abstract |
| `var AnimalList oList.` | 20519 generic type … without type arguments |

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# One Class, One R-code

### One .r per generic type

However many `<Cat>`, `<Dog>` clients use it

Name = file name, as always

### 12 → 13: no recompile

Progress: “Recompilation is not required between 12 and 13”

Generic r-code needs 13.0: 12.8 crashes

12.8.11 running 13.0 r-code ends at the first generic object, no ABL error

### 13.0 is an innovation release

12.8 is the LTS, so generic types will not be available

**Design with generics now, ship them when your sites run 13**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Plays Well with Progress.Collections

## Wrap them

`AnimalList<T>` keeps a `List<T>`

`GenericDictionary<T>` keeps a `HashMap<String, T>`

## Implement them

```
class Person implements
IComparable<Person>
```

for a `SortedSet<Person>`

## Nest them

`List<Result<Customer>>`

`HashMap<String, AnimalList<Cat>>`

## Tighten them

HashMap keys must implement `IHashable`, checked when you add one

`K as IHashable` checks at compile time

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# .NET Generics: Same Brackets, Different Animal

| | .NET generic types | ABL user-defined generics |
|---|---|---|
| **Since** | 10.2A (12.8 on Linux) | 13.0 |
| **You can** | Use constructed types | Define your own |
| **Type arguments** | .NET types and ABL primitives (`List<CHARACTER>`); not ABL classes | ABL classes and interfaces only |
| **Inherit / implement in ABL** | No | Yes |
| **Mixing** | — | No .NET types as T |

**Same angle brackets, different rules: .NET takes primitives, ABL takes objects**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Live Demo 2: What Does the Runtime Know?



### Before

- Is a generic object at runtime just an object?

» Code:

```
oCls = oCats:GetClass().
message oCls:IsGeneric().
oArgs = oCls:GetTypeArguments().
message oArgs[1]:TypeName.

oAny = dynamic-new
  "…AnimalList<…Dog>" ().
```

### You'll see

- Type name, arguments, parameters
- A Dog made by reflection
- Gold refused at runtime

**The runtime knows the type arguments. Dynamic code gives up the compile-time check.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Agenda

| | | |
|---|---|---|
| 1 | **Why?** | The same pattern, built again and again |
| 2 | **How?** | Classes, interfaces, constraints |
| 3 | **The rules** | What you can and can't do with T |
| 4 | **Under the hood** | Compiler, runtime, r-code, interop |
| 5 | **In the real world** | SmartComponent Library lists and dictionaries, and friends |
| 6 | **Gotchas** | And when generics are the wrong tool |

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# The SmartComponent Library Way

**One of 131 lists**

```
class ListNameValuePair
  inherits GenericList:
  {Consultingwerk/Framework/
    Base/GenericList.i
    NameValuePair}
end class.
```

**One of 36 dictionaries**

```
class DictionaryIClassCache
  inherits GenericDictionary:
  {Consultingwerk/Framework/
    Base/GenericDictionary.i
    IClassCache}
end class.
```

**The include writes the typed Add() and GetItem(). Dictionaries: a CAST inside.**
**Lists on 12.5+: a native List<T> inside.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# {1} Is Not a T

**GenericList.i (excerpt)**

```
method public {1} GetItem (
    input piIndex as integer):
  &IF DEFINED (NativeLists) EQ 0 &THEN
  return cast (super:InternalGetItem (
    piIndex), {1}) .
  &ELSE
  return this-object:NativeList:Get(
    piIndex).
  /* catch: InvalidListIndexException */
  &ENDIF
end method.
```

**Preprocessed for ListNameValuePair**

```
/* 13.0 build: NativeLists defined */
method public NameValuePair GetItem (
    input piIndex as integer):
  return this-object:NativeList:Get(
    piIndex).
  catch err as Progress.Lang.Error: …
end method.

/* 11.x - 12.4 builds */
  return cast (super:InternalGetItem (
    piIndex), NameValuePair) .
```

**An include is copied into every class. A generic type is compiled once.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# GenericDictionary<T>: One Class, No Include

13.0+

GenericDictionary.cls (shortened)

```
using OpenEdge.Core.String.
using Progress.Collections.*.

class Demo.Collections.Generic.GenericDictionary<T as Progress.Lang.Object>:
  define private variable mItems as HashMap<String, T> no-undo.

  method public T Add(input pKey as character, input pItem as T):
    mItems:Add(new String(pKey), pItem).
    return pItem.
  end method.

  method public T GetItem(input pKey as character):
    return mItems:GetValue(new String(pKey)).
  end method.
```

**Same core API as GenericDictionary.i: Add, AddWhenNotContained, GetItem, GetItemOrNull …**
**Plus a typed ListChanged event. No Progress.Lang.Object, no CAST.**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Keep the Names, Lose the Include? Not Yet

**Before (any release)**

```
class DictionaryCat
  inherits GenericDictionary:
  {Consultingwerk/Framework/Base/
     GenericDictionary.i Cat}
end class.
```

**After (13.0+)**

```
class DictionaryCat
  inherits GenericDictionary<Cat>:
end class.
```

**Compiles, but unusable in 13.0.0 (20498, 20508, 14285)**

### 1. Write it once

The generic base class

### 2. Keep the names

Once Progress fixes non-generic subclasses

### 3. New code

Uses the generic type directly

**Today: new code uses GenericDictionary<Cat> directly**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Bye-bye CAST: Result<T>

Result.cls (outline)

```
class Result<T as Progress.Lang.Object>:
  define public property IsSuccess as logical …
  define public property Value as T …
  define public property ErrorMessage as char …

  constructor public Result(input pValue as T):
  constructor public Result(input pError as char):

  method public T GetValueOrThrow():
end class.

oCust = oRepo:FindById("2"):GetValueOrThrow().
```

## No ambiguity

T can never be CHARACTER, so the two constructors never collide

## No shortcuts

No `Result:Ok()` (no statics)

No `Map<U>()` (no generic methods)

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Live Demo 3: The Dictionary, and Friends



## Before

- One subclass per type
- `CAST` inside every `GetItem`

» **Code:**

```
var GenericDictionary<Cat> oCats
  = new GenericDictionary<Cat>().
oCats:Add("puschel",
          new Cat("Puschel")).
oCats:GetItem("puschel"):Name.

var ListRepository<Order> …
```

## You'll see

- Typed GetItem, no CAST
- Typed events
- Result and repository

**One generic class instead of one class per type**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Agenda

| | | |
|---|---|---|
| 1 | **Why?** | The same pattern, built again and again |
| 2 | **How?** | Classes, interfaces, constraints |
| 3 | **The rules** | What you can and can't do with T |
| 4 | **Under the hood** | Compiler, runtime, r-code, interop |
| 5 | **In the real world** | SmartComponent Library lists and dictionaries, and friends |
| 6 | **Gotchas** | And when generics are the wrong tool |

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# What Could Possibly Go Wrong?

### Exact type arguments

`List<Cat>` is not a `List<Animal>`

### Overloads vs T

`Add(T)` + `Add(Cat)`: ambiguous for `<Cat>`

### Non-generic subclasses

`class CatList inherits AnimalList<Cat>`: compiles, can't be used (13.0.0)

### T trusts its constraint

`return new Dog()` compiles in a `T` method; no runtime check

### Docs vs compiler

ABSTRACT: docs yes, compiler no. CATCH on T: the other way round

**Know them before the first code review, not after**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# When Generics Are the Wrong Tool

### Only one type, ever
Write the class for that type

### Behaviour differs per type
That's polymorphism: use an interface

### Data, not objects
Temp-tables, datasets, handles, primitives can't be T

### You need statics
Factories and singletons: not in a generic class

### NEW T() everywhere
Reflection for every object: rethink the design

### Your sites run 12.8
Wait for 13 in production

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# In Closing

**Write it once, let the compiler check it everywhere**

- One generic class instead of 167 generated ones (once subclassing works)

**Start where you copy-paste, or CAST**

- Collections, results, repositories, caches, event payloads

**T is its constraint, both ways**

- No NEW T, no statics, no abstract, no generic methods: design around them

**13.0 today, your production sites tomorrow**

- Innovation release now; wait for 13 on your LTS sites

**Strong typing is (still) your Best Friend**

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Get the Code



## Samples on GitHub

- Every class and demo from today
- One documented compile error per file
- Open questions as experiments
- MIT licence



[github.com/consultingwerk/ABL-User-Defined-Generics-Samples](github.com/consultingwerk/ABL-User-Defined-Generics-Samples)

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# My Sessions at PUG Challenge Amsterdam

| Session | Day | Time | Room |
|---|---|---|---|
| .NET on Linux. Why bother? | Wed 4 | 3:00 – 4:00 PM | New York 3 |
| **What are user-defined generic types and why should I use them?** | **Thu 5** | **10:15 – 11:15 AM** | **New York 3** |
| Managing secrets in OE configurations | Fri 6 | 1:45 – 2:45 PM | New York 3 |

*Highlighted: this session*

All sessions: [pugchallenge.org](pugchallenge.org)

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Other Sessions Worth Your Time

| Session | Speaker | Day | Time | Room |
|---|---|---|---|---|
| Stop Mocking Everything: Testing OpenEdge with Real Databases in Disposable Environments | Julian Lyndon-Smith | Thu 5 | 11:45 AM – 12:45 PM | Brussels 1 |
| OpenEdge Platform and Language (Birds of a Feather) | Raphael Ross | Thu 5 | 4:30 – 5:30 PM | Athens |
| Regular Expressions for ABL Developers | Sebastian Wandel | Fri 6 | 10:15 – 11:15 AM | New York 2 |

All sessions: [pugchallenge.org](pugchallenge.org)

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.

# Questions

© 2026 Consultingwerk Application Modernization Solutions Ltd. All rights reserved.


Consultingwerk
software architecture and development