

# How... to insert a column into a listbox

## Part 1: How does inheritance work in ClassiX®?

### Requirements

- ClassiX® system installed
- Codewright or another text editor
- knowledge of:

basic ClassiX® design conventions classes inheritance module	widget action list event message variables
--	--

### Task:

In the last “cooking recipe”, we have learned how to call the monitor window, and how to modify a module. Now we want to insert a column into the application listbox.

### Content

- ObjectListView
- Implementation
- What has happened?
- How does inheritance work?
- Where can I modify code?

ObjectListView, OListView, OLView

A listbox gets defined with **ObjectListView** in InstantView®. OlistView and OLView are accepted abbreviations.

### Syntax

ObjectListView(name~aliasName, flags, x, y, w, h)

#### Parameter:

Name	*	identifier or CLASS:: <b>ausdr</b>
aliasName		an additional identifier
Flags		flags
X	*	position X (in Minicells)
Y	*	position Y (in Minicells)

W	*	width (in Minicells)
H	*	height (in Minicells)

\* - obligatory parameters

The action list and the event INITIALIZE control the display:

### Syntax

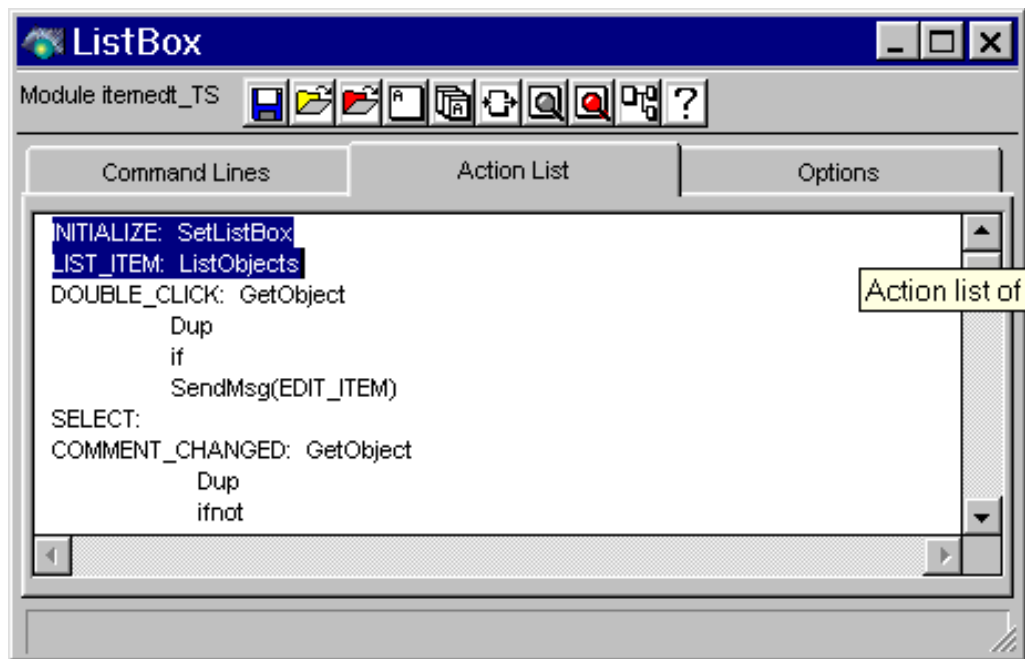
```
[ INITIALIZE : SetListBox ]
```

```
Define(SetListBox);
```

defines the listbox content. The function SetListBox is defined further above in the module or in a basic module of the module.

### Implementation

Open ClassiX® and clicked on the domain „Teilestamm“. Then open the listbox monitor window and call the Reiter “action list”.



INITIALIZE : We find SetListBox here, but we neither find the expected ObjectListView "ListBox" with the action list with INITIALIZE nor the function SetListBox in the code of itemedt\_TS.mod.

### What has happened?

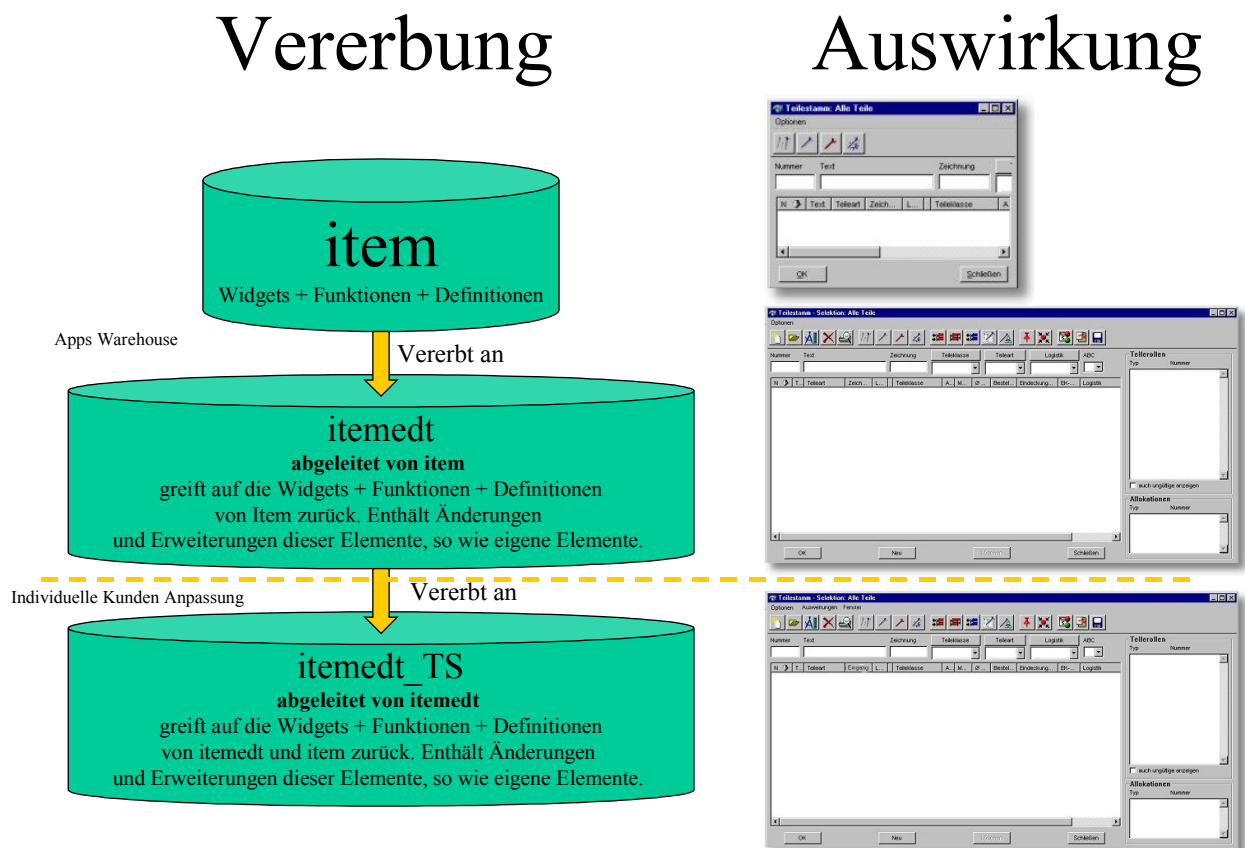
The standard toolbar displays “itemedt\_TS” for the module name. We know from

the how-to conventions, that modules, which end with an underscore and a code in capitals, are derived customer modules.

Therefore, the accessed module is an individual customer module, which has been derived from an already defined module: The listbox code can be found in a basic module.

## How does inheritance work?

Deriving a new module from another one, the older one is the basic module for the new one. The new module inherits the entire code. It is basically the basic module's clone, without needing to the entire code again. Code modifications in the derived module don't change the basic module, whereas changing the basic module will automatically affect all derived modules. That way, very different modules can be derived from the same basic module. All elements are adopted from the basic module, as long as they are not changed in the derived module or overwritten by an element with an identical name.



For the elements, this means in detail:

## variable

If the variable `x` exists in the basic module, the derived module will also contain a variable named `x`. Changing the value for `x` in another derived module has no effect on `x`.

## statement

If a statement `s` has been declared in the basic module, this statement will also be known in the derived module, unless a new variant of `s` has been defined here. Since such a redefinition is often an upgrade of the old statement, it is possible to refer to the according basic module statement with

**moduleName::s or super::s.**

whereas `super::s` should be used always!

## action list

The basic module action list is also always inherited. The reaction to a specific event can simply be redefined in the derived module. The action caused in the basic module, gets replaced by overwritten one in the derived module. The overwritten statements can be activated with [SendMsg\(, MESSAGENAME\)](#) or [SendMsg\(,SUPER\)](#).

## Windows

Basic module windows also become part of the derived modules. If the derived module contains a window with the same name, this window inherits all the child objects from the basic window which don't have an equivalent with the same name.

## Where can I modify code?

The super-user can only modify code in customized modules. If the module has never been modified before (therefore it has never been derived, either), a new, customized module has to be derived. **AppsWarehouse® modules must not be modified by the super-user.**

## Abstract:

We have learned how to define a listbox. To modify the code, we maintained fundamental knowledge of inheritance in ClassiX® and a rule, which defines, what code may be modified by a super-user and which code mustn't.

## Further information

**Part 2 – How to find and modify code**

**Part 3 – How to find a data field**

**Part 4 – Complex data types!**

**Part 5 – Listbox **cosmetics****

**Part 6 – How to find /modify a column head line**