

**NAME**

XtCreateWidget, XtVaCreateWidget, XtCreateManagedWidget, XtVaCreateManagedWidget, XtDestroyWidget – create and destroy widgets

**SYNTAX**

```
#include <X11/Intrinsic.h>
```

```
Widget XtCreateWidget(const char *name, WidgetClass widget_class, Widget parent, ArgList args, Cardinal num_args);
```

```
Widget XtVaCreateWidget(const char *name, WidgetClass widget_class, Widget parent, ...);
```

```
Widget XtCreateManagedWidget(const char *name, WidgetClass widget_class, Widget parent, ArgList args, Cardinal num_args);
```

```
Widget XtVaCreateManagedWidget(const char *name, WidgetClass widget_class, Widget parent, ...);
```

```
void XtDestroyWidget(Widget w);
```

**ARGUMENTS**

<i>args</i>	Specifies the argument list to override the resource defaults.
<i>name</i>	Specifies the resource name for the created widget, which is used for retrieving resources and, for that reason, should not be the same as any other widget that is a child of same parent.
<i>num_args</i>	Specifies the number of arguments in the argument list.
<i>parent</i>	Specifies the parent widget.
<i>w</i>	Specifies the widget.
<i>widget_class</i>	Specifies the widget class pointer for the created widget.
...	Specifies the variable argument list to override the resource defaults.

**DESCRIPTION**

The **XtCreateWidget** function performs much of the boilerplate operations of widget creation:

- Checks to see if the `class_initialize` procedure has been called for this class and for all superclasses and, if not, calls those necessary in a superclass-to-subclass order.
- Allocates memory for the widget instance.
- If the parent is a subclass of **constraintWidgetClass**, it allocates memory for the parent's constraints and stores the address of this memory into the constraints field.
- Initializes the core nonresource data fields (for example, parent and visible).
- Initializes the resource fields (for example, background\_pixel) by using the resource lists specified for this class and all superclasses.
- If the parent is a subclass of **constraintWidgetClass**, it initializes the resource fields of the constraints record by using the constraint resource list specified for the parent's class and all superclasses up to **constraintWidgetClass**.
- Calls the initialize procedures for the widget by starting at the **Core** initialize procedure on down to the widget's initialize procedure.
- If the parent is a subclass of **compositeWidgetClass**, it puts the widget into its parent's children list by calling its parent's `insert_child` procedure. For further information, see Section 3.5.
- If the parent is a subclass of **constraintWidgetClass**, it calls the constraint initialize procedures, starting at **constraintWidgetClass** on down to the parent's constraint initialize procedure.

Note that you can determine the number of arguments in an argument list by using the **XtNumber** macro. For further information, see Section 11.1.

The **XtCreateManagedWidget** function is a convenience routine that calls **XtCreateWidget** and

**XtManageChild.**

The **XtDestroyWidget** function provides the only method of destroying a widget, including widgets that need to destroy themselves. It can be called at any time, including from an application callback routine of the widget being destroyed. This requires a two-phase destroy process in order to avoid dangling references to destroyed widgets.

In phase one, **XtDestroyWidget** performs the following:

- If the `being_destroyed` field of the widget is **True**, it returns immediately.
- Recursively descends the widget tree and sets the `being_destroyed` field to **True** for the widget and all children.
- Adds the widget to a list of widgets (the destroy list) that should be destroyed when it is safe to do so.

Entries on the destroy list satisfy the invariant that if `w2` occurs after `w1` on the destroy list then `w2` is not a descendent of `w1`. (A descendant refers to both normal and pop-up children.)

Phase two occurs when all procedures that should execute as a result of the current event have been called (including all procedures registered with the event and translation managers), that is, when the current invocation of **XtDispatchEvent** is about to return or immediately if not in **XtDispatchEvent**.

In phase two, **XtDestroyWidget** performs the following on each entry in the destroy list:

- Calls the destroy callback procedures registered on the widget (and all descendants) in post-order (it calls children callbacks before parent callbacks).
- If the widget's parent is a subclass of **compositeWidgetClass** and if the parent is not being destroyed, it calls **XtUnmanageChild** on the widget and then calls the widget's parent's `delete_child` procedure (see Section 3.4).
- If the widget's parent is a subclass of **constraintWidgetClass**, it calls the constraint destroy procedure for the parent, then the parent's superclass, until finally it calls the constraint destroy procedure for **constraintWidgetClass**.
- Calls the destroy methods for the widget (and all descendants) in post-order. For each such widget, it calls the destroy procedure declared in the widget class, then the destroy procedure declared in its superclass, until finally it calls the destroy procedure declared in the Core class record.
- Calls **XDestroyWindow** if the widget is realized (that is, has an X window). The server recursively destroys all descendant windows.
- Recursively descends the tree and deallocates all pop-up widgets, constraint records, callback lists and, if the widget is a subclass of **compositeWidgetClass**, children.

**SEE ALSO**

**XtAppCreateShell(3)**, **XtCreatePopupShell(3)**  
*X Toolkit Intrinsics – C Language Interface*  
*Xlib – C Language X Interface*