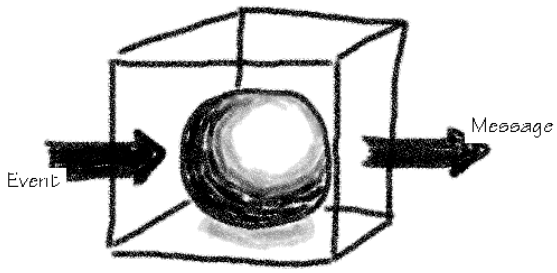
# Chapter 6 Links

## General Overview

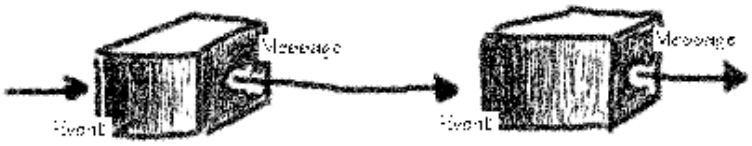
### Some Background Considerations

Defining a behavior for components in mindmap is equivalent to the process of programming an application in a conventional development environment such as C++, VisualBasic, PowerBuilder, etc. In mindmap, defining the behavior is accomplished by assigning one or more links to a component. A link is comprised of exactly one incoming event and one outgoing message. This pair is what we call a link. The process of defining such links in mindmap is referred to as linking components.



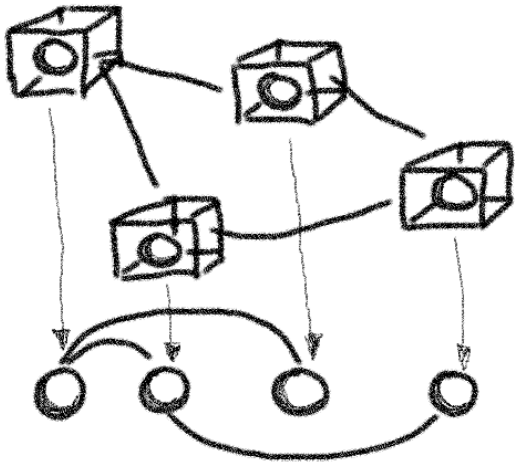

*mindmap*'s event - message model

An event functions as a trigger for a component. When a trigger occurs, the component which has been associated with the trigger will receive control and do as it has been instructed in the message. It will generate the message. A message, in turn, will cause a specific reaction by the associated component or it will act as an event for a subsequent component.



One component passing a message to another component

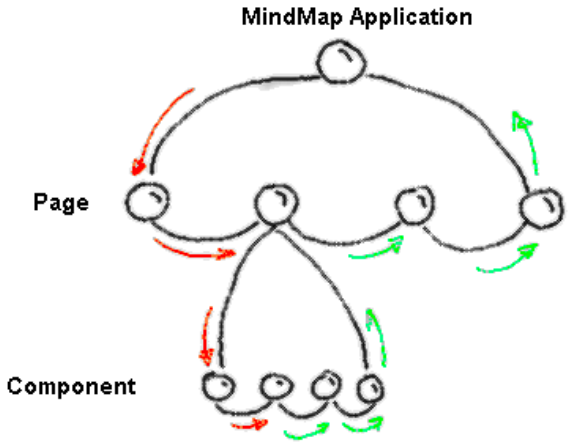
Thus, building an application can be viewed as constructing a network of components, sending and receiving both events and messages. In the above model, a component is receiving exactly one event and generating one message. In other words, it has exactly one link associated with it. If this were always the case, an application would be fairly limited. Therefore, mindmap allows multiple links to be assigned to a component.



A network of *mindmap* components

In a two-dimensional representation, the cubes represent the components and are displayed as a network, albeit in this case a very simplistic one. If the two-dimensional representation is collapsed into one dimension (represented by the balls), then an application can be viewed as a single list of components connected in a parent-child relationship.

This leads to a very important aspect of mindmap, which should be kept clearly in mind when constructing applications — the mechanism by which messages are passed in the system. Let us take this, and granted, a somewhat abstract representation, one level further. Consider the balls in the above representation to be pearls on a chain. In this little example, the application would consist of one string of pearls. Let's look at a somewhat more complex application.




Message chain

This is a necklace of necklaces — where multiple strings of pearls are interconnected. The top level of this picture represents the application itself. At this level, events are received from the operating system. When a user clicks on the mouse, the operating system intercepts the event and determines which programs are running at the time. It then determines which program is in foreground and should presumably be waiting for user interaction. Once this has been determined, the operating system forwards the event into the message handling facility of the program, in this case mindmap.

Next, mindmap – actually its event-message handler – grabs the event and forwards it to the first page. All pages are on the same string of pearls, meaning that they are logically at the same level. The event is passed into the necklace and the first page receives it. The page determines whether or not it is to act

on it. In case it decides it doesn't want to deal with it, it passes the event on to the next page on the necklace. The process begins anew. Eventually either a page decides to process the event or it ends up back at the top of the chain, in which it is simply discarded.

If a page decides that it should deal with the event, it grabs the event, opens up its little necklace of components on the page and forwards the event into this chain. The process begins again. The first component on the page takes a look at the event and decides if it wants to act on it. If not, it passes the event on to the next in line. Either the event is squeezed back up to the page level without having been dealt with, or some component on the page has grabbed it and acted on it. In this case, the event will be flagged so that subsequent components will not deal with it. The flagged event is then passed through the component level of the chain, up to the page level, where it is then rushed back up to the application level.

The sequence for processing events is – graphically speaking – either left to right or right to left, depending on the particular type of event. Well, how does this relate to the components as viewed on the screen? Easy. The first page in a mindmap application is the leftmost page. The next created page will be 'next in line', or to the right of the previous page. Inserting a new page can be envisioned as cutting open the necklace, adding in a new pearl and knotting the chain back together again.

The same metaphor can be used for the components themselves. The first placed component is first-in-line. It is at the leftmost position on its segment of the necklace. The next created component is attached to the right of its predecessor.

This 'pearls-on-the-necklace' order can also be viewed differently; namely, in the sense of foreground and background. The first component placed on a page is in the background. The next component is in foreground, relative to the first, and so on. Thus, the last placed component is in foreground, relative to all other components on the page. If you now move a component into background (via the menu option **Edit | Background** or vice versa **Edit | Foreground**, see page 67), what you are actually doing is rearranging the pearls on the necklace. You are, figuratively speaking, cutting open the necklace and inserting the component into a new position.

Applying this metaphor to groups, we get the following picture: When you create a group of components, what is really happening is that the component necklace segment is cut open, a new type of component is created and inserted (being the non-visible group component) and all components belonging to the group are strung onto the new segment, which dangles off the component segment. A group of groups is, thus, a necklace of necklaces.

Keep in mind, that the order in which the components receive and process their messages, also corresponds to the order in which they are painted onto the screen -- from background to foreground or from foreground to background.

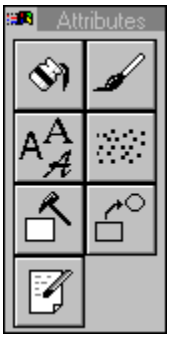
The direction of event passing actually depends on the type of event. Generally, all mouse-related events are passed from right to left (foreground to background) in order to make sure that events being in the foreground receive mouse clicks first, as one would expect. On the other hand, system related events like `Application started` or `F3 key pressed` are passed from left to right.

We strongly recommend that links acting on the same type of system related event (like `Application started` or `Page activated`) are kept together on a single component. This makes it easiest to keep control over the sequence of operation.

## How to Define a Link

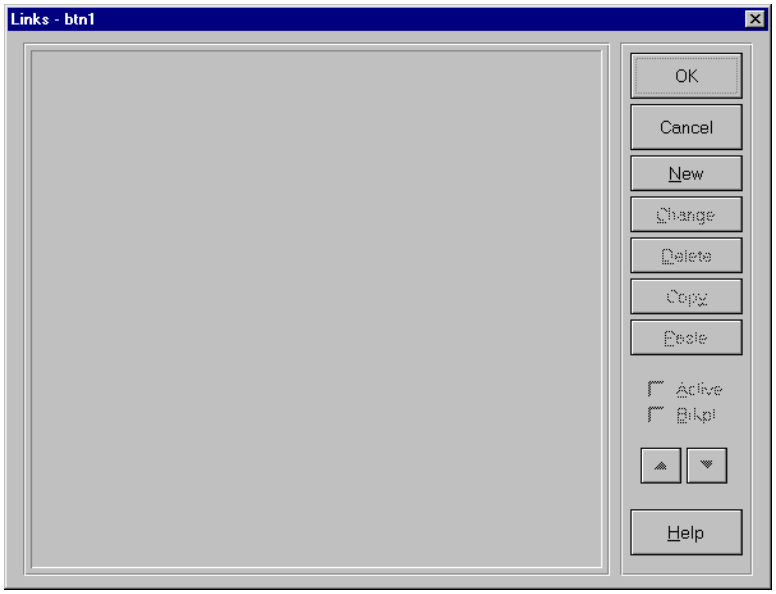
Before getting into the details of what each event and message actually does, let's look at how they are assigned to a component.

Select a component to which you want to assign a link. Next, either click on the menu option **Properties | Links**, press the **F6** function key, or activate the attribute toolbox for the component. Let's assume you did the latter.



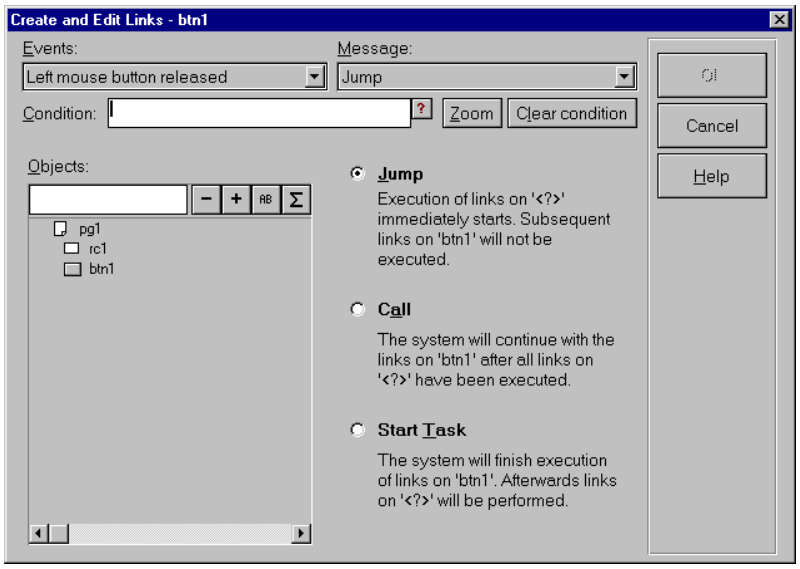

Attributes of a component

Now, click on the icon displaying an arrow leaving a little square and pointing towards a little circle. This will result in the following screen being displayed:




This dialog will list all links that are defined for a selected component

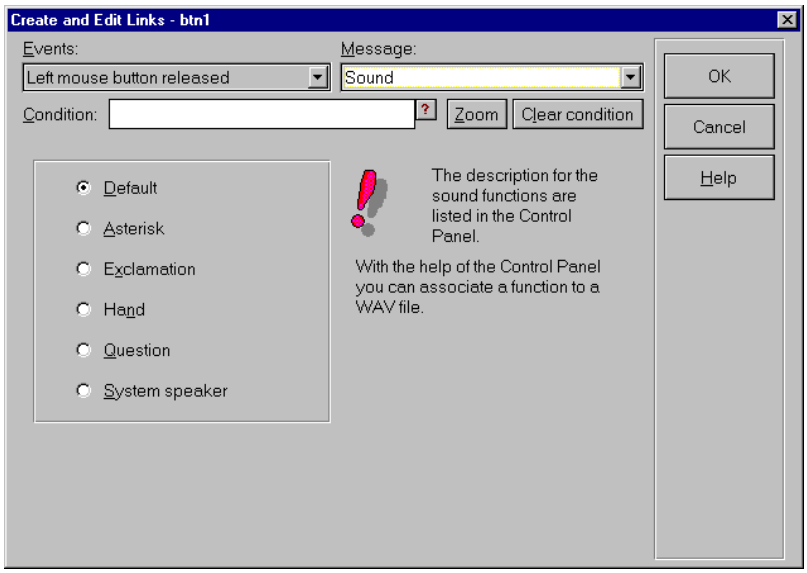
Since no links have been defined so far, the major portion of this dialog box is empty. In order to define a new link, click on the New button on the right side of the dialog box. This will open up the following dialog box:




Multipurpose dialog box to create and modify links

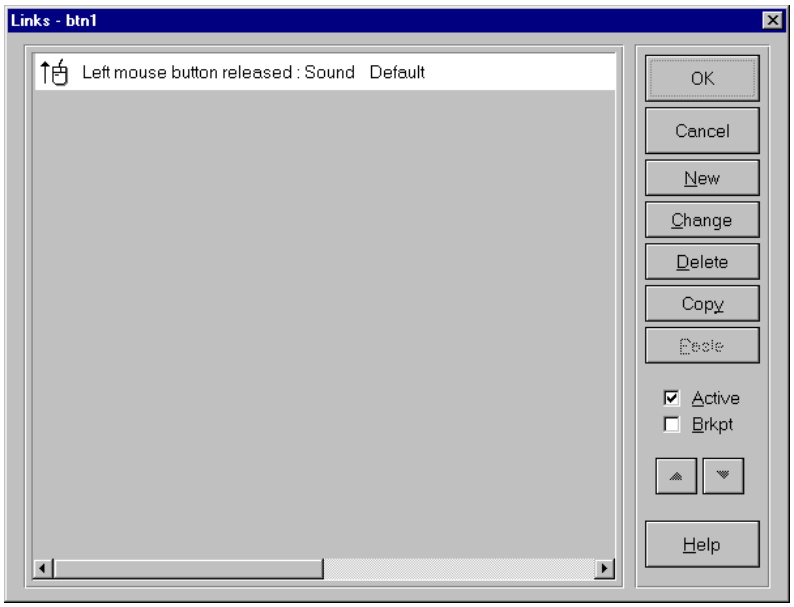
This is the control center. Here is where you actually define the behavior of a component. At the top left of this screen, you see the Events list. Next to it is the Message list. Underneath both lists is an area which deals with conditions. The rest of the screen is dedicated to whatever definitions are necessary, in conjunction with the selected event-message combination. Therefore, it will change its appearance depending on the selection made earlier.

To demonstrate the method by which a link is defined, drop down the Events list. Select one of the events, such as Left mouse button released. Next, drop down the Message list and select the message entitled Sound. This will result in the following screen:




A different type of link - play a sound

As you will notice, the bottom section of the dialog box has changed to reflect the selection of the Sound message. Click on any one of the options on the left side and accept the selection by clicking on the OK button on the right side of the screen. This will result in returning you to the initial link dialog box, only this one has now been updated to reflect the new link definition.

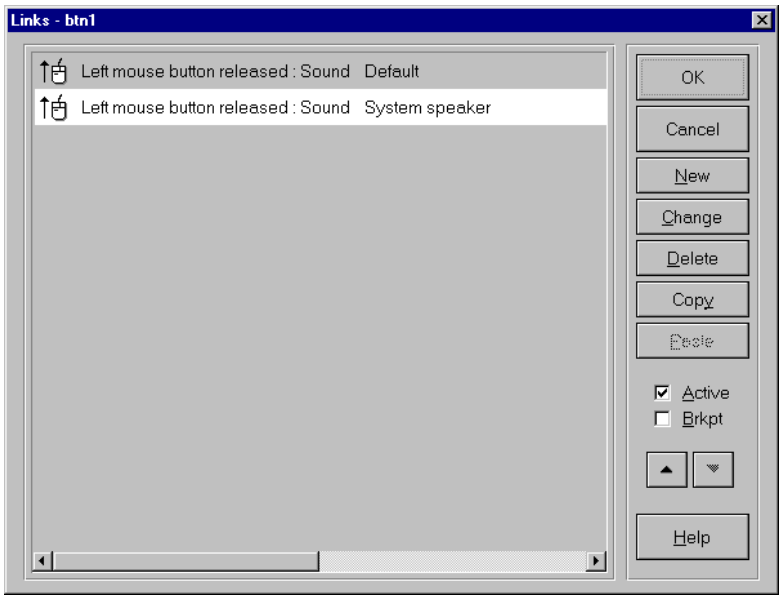



A link has been created for the component btn1

The dialog box now displays a link defined for the component **btn1** (see the caption bar on the dialog box). The link can be read to say that:

'When the mouse has been moved on top of the component btn1, and the left mouse button has been clicked and subsequently released, the sound associated with the default setting will be produced.'

Next, create another link, this time associating a different sound with the same event. The screen should then look similar to this one:




Two links are defined for the button btn1

If you were to execute this little application, then the following actions would take place. Once you clicked on the push button named btn1, the sound associated with Default would be played (provided you have a sound card installed), immediately followed by the sound associated with System speaker. In other words, the processing sequence of links is top-to-bottom of the list for any given component. There is one aspect you should be aware of, though. Some events are generated sooner than other events, an example being Application started. If you have assigned this event to any component in the application, then it will be executed, and thus the message associated with it will be performed, before the list of any component will be processed.

Relative to a component, the link list is processed from top to bottom. Relative to the application though (which in some cases might override the apparent sequence associated with any given component), certain events are generated without user interaction, and prior to any user interaction events. These will be processed first.

If you wish to rearrange the order of link processing, you can easily do so. In this simple case, let's assume you want to hear

the sound associated with the System speaker entry before the Default sound is generated. In this case, select the second link (Sound Speaker sound). It will invert the background color. On the bottom right hand side of the dialog box, you can see two arrow heads, one pointing up, the other pointing down. Click on the one pointing up. This will cause the second link to be moved up, in front of the Sound Default link.

The mouse cursor indicates an insertion point

Another way of rearranging the links is by selecting the link you wish to move. Then, keeping the Left mouse button pressed, drag it to the position you wish it to be moved to. The cursor will change its appearance to display where it will accept the placement of the link. This cursor will appear whenever the mouse has been positioned between two existing links, or at the top or the bottom of the list of links.

You can also delete, copy, and paste links within this screen, as well as between links screens.

If you wish to delete a link, select it and then click on the Delete button on the right side of the dialog box. If you want to delete multiple links, you can select them either individually by keeping the **CTRL** key pressed and clicking on the link, or by keeping the **SHIFT** key pressed and clicking on the links. Using the **CTRL** key approach permits you to select non-contiguous links. Using the **SHIFT** key method will select all links between the first and the last selection made using the mouse.

Using the Copy button on the right side of the dialog box, you can place the selected link(s) on the clipboard. You can employ the same selection methods just described for the delete function. Once selected, the links are placed on the clipboard as soon as you click on the Copy button. Copying links to the clipboard will overwrite any links placed there previously.

You can retrieve the links from the clipboard by clicking on the Paste button. The link(s) off the clipboard will be placed immediately above the links selected at the time. Thus if you already have to links on the screen and you insert the contents of the clipboard, then assuming the first of the two links is selected, the contents will be placed as the first links.

Another feature available is that of toggling a link active and inactive. Links are active by default. If you wish to temporarily turn one off (make it inactive), then simply select it and mark

the check box on the lower right hand side of the dialog box. The selected link(s) will be grayed and will not be executed during run time.

# Events

## Common Events

All components have a common heritage in regards to the events they support. When a component developer builds a new component he/she is offered a set of generic events. It is then the decision of the developer to support all, or any sub-set, of these events. In addition, the developer can then register new events, that are specific to the component being developed.

The simpler components do not support their own events. They merely inherit their events from the generic list of events. Following is the set of generic events. In general, all – or at least a major portion of them – are supported by all components:

Many users have difficulties performing mouse double click operations. This is in part due to the sensitivity settings for the mouse. When attempting to *double click* many users move the mouse a few pixels. This is often enough for Windows to interpret the action as two separate single mouse clicks. It is generally advised to avoid using this event, if at all possible.

| Icon | Event | Description (When generated) |
|---|---|---|
|  | Left mouse button clicked | When left mouse button is pressed and before it is released. |
|  | Left mouse button released | When left mouse button has been released.<br>This (and Right mouse button released) is the preferred event on which to trigger buttons. |
|  | Left mouse button double clicked | When left mouse is clicked twice in rapid sequence. |
|  | Right mouse button clicked | When right mouse button is pressed and before it is released. |
|  | Right mouse button released | When right mouse button has been released. |

| Icon | Event | Description (When generated) |
|---|---|---|
| | | This (and Left mouse button released) is the preferred event on which to trigger buttons. |
|  | Right mouse button double clicked | When right mouse button is clicked twice in rapid sequence. |
|  | Activate Page | When control is passed to current page (jumped to or to component on page).<br>Use this event when you wish actions to be taken before anything else on the page, to which is being jumped, is to be executed. |
|  | Deactivate Page | When control is passed to another page.<br>Use this event if you need to 'clean up' something, before turning control over to the next page. |
|  | Mouse movement | Any movement of the mouse will generate this event. This event is sent to the component which the mouse is currently above.<br>Use this event very cautiously, since it will be generated anytime the mouse is moved within the border of the selected component. |
|  | Mouse movement into object | When the mouse cursor is moved into selected component.<br>Use this event in conjunction with the changing of the appearance of the mouse cursor. |
|  | Mouse movement out of object | When the mouse cursor leaves the selected component.<br>Use this event in conjunction with the changing of the appearance of the mouse cursor. |
|  | Application started | When the application is started (put into run mode or executed as EXE file). |

| Icon | Event | Description (When generated) |
|---|---|---|
|  | Application terminated | When the application is normally terminated (put into edit mode or quit as EXE file), this event will be generated.<br>This event is often used to clean house (i.e., set variables to some predefined value) before leaving the application. |
|  | Goal of a jump | When the selected component has been jumped to (used in conjunction with sub routines/ sub assemblies). |
|  | Error in calculation | When an error in evaluating a formula is encountered. This event is sent to all components on the active page.<br>Parser error messages are described on page 390. |
|  | Begin Drag&Drop | When a drag&drop operation using the mouse begins. This event is sent to the component initiating the drag&drop operation. |
|  | End Drag&Drop | When a drag&drop operation using the mouse ends. This event is sent to the component that has initiated the drag&drop operation. |
| | **F1**-**F12**-Keys | When any function key is pressed in run mode for the selected component. This event is sent to all components on the currently active page.<br>**F4** toggles back into edit mode, if the file is running as an .MM file. When running an .MM file which has been converted to an .EXE file, this function key will not toggle between edit mode and run mode. |
|  | **Esc**-Key | When the **ESC** key is pressed for the selected component. |

# Component Specific Events

The following list contains those events which are available if, and only if, the associated component has been installed. These events are specific to the particular component. They will only be visible in the list of events for a component of the same type.

## Buttons

| Icon | Event | Description (When generated) |
|---|---|---|
|  | Button pressed | When a command button has been pressed and before it is released. |
|  | Button released | When a command button has been released. |
|  | Row up/left | When Up arrow has been clicked on a selected scroll bar component. |
|  | Page up/left | When area on selected scroll bar between elevator and the Up arrow has been pressed. |
|  | Row down/right | When Down arrow has been clicked on a selected scroll bar component. |
|  | Page down/right | When area on selected scroll bar between elevator and Down arrow has been clicked. |
|  | Move | Only when the elevator bar on the scroll bar is moved up and down (or left and right on a horizontal scroll bar), this event will be generated. |
|  | Scroll change | The event is generated when either the elevator bar is moved, one of the arrows has been clicked on, or the area between the elevator bar and an arrow has been clicked. |

➤ Read more about buttons on page 125.

## Data Tables

| Icon | Event | Description (When generated) |
|---|---|---|
|  | Receive focus | Whenever the cursor is placed inside a cell of the data table, this event will be generated. This event is only generated when the data table itself receives focus. Use the event Cursor changed if you want to determine each single click into a cell of the data table. |
|  | Lose focus | Whenever the data table itself or a cell inside the data table loses focus, this event will be generated. |
|  | Cursor changed | Anytime the cursor is moved to another cell within the data table, this event is generated. |
|  | Double-click | Whenever a cell in the data table receives a double click, this event is generated. |
|  | Abort edit mode | This event is sent when the cell edit mode is canceled with the **ESC** key. |
|  | Begin edit mode | The normal procedure to begin the edit mode in the cell of a data table is to press the **CTRL+E** key. This will generate the event. If the cursor is placed inside a cell of the data table and the user begins to enter in characters, the event will also be generated. |
|  | Cell changed | Whenever the content of a cell has been changed, this event will be generated. Keying in the same characters as are already in the cell, will not generate this event. |

| | | |
|---|---|---|
|  | Order changed | If the attribute, which permits the user to rearrange the rows at run time, has been set, then the event will be generated whenever a row has actually been rearranged. The process for rearranging rows is to select a row, keeping the Left mouse button pressed, moving it to another row and letting go of the left mouse button. The row can only be 'dropped' when the cursor changes its appearance, signaling a valid position. |

➤ More information about data tables can be found on page 184.

## Input Fields

| Icon | Event | Description (When generated) |
|---|---|---|
|  | Enter | If the cursor is inside the input field (input field has focus) and the user presses the **ENTER** key, this event is generated. If the input field is set to display multiple lines, then **ENTER** will cause carriage/line feed and the cursor will be placed into the first column of the next row. |
|  | Keyboard input | If the cursor is inside the input field (input field has focus) and the user begins to type in characters this event is generated. |
|  | Receive focus | Whenever the cursor is placed inside the input field, this event is generated. |
|  | Lose focus | If the input field had focus and the cursor was used to click somewhere other than inside the input field, this event is generated. |

| | | |
|---|---|---|
|  | Cursor changed | If the cursor is inside the input field (input field has focus) and the user moves the cursor (caret) to another position by clicking on the arrow keys on the keyboard, this event will be generated. Moving the position of the caret by means of the mouse will not generate this event. |

➤ The input field is described on page 193.

## MCI

| Icon | Event | Description (When generated) |
|---|---|---|
|  | Mode changed | This event is generated, if the mode of the multimedia component changes (e.g. playing, seeking, stopped, ...). |
|  | Device not ready | This event is generated, if the device is in a state in which playing a multimedia file is not possible, as is the case when a CD was ejected from a CD drive. |
|  | Stopped | This event is generated, when the device ends playing a multimedia sequence, regardless of whether the device stops through user interaction or if the end of the media is reached. Prior to sending this event, the Mode changed event will be sent. |
|  | Playing | This event is generated, when the device starts playing a multimedia sequence. The event is sent after the Mode changed event has been generated. |
|  | Recording | This event is generated, when the device starts recording a multimedia sequence. The event is sent after the Mode changed event has been |

| | | |
|---|---|---|
| | | generated. |
|  | Seeking | This event is generated, when the device moves to a new position of a multimedia sequence. The event is sent after the Mode changed event has been generated. |
|  | Paused | This event is generated, when the device is in pause mode. The event is sent after the Mode changed event generated. |
|  | Device is open | This event is generated, when a multimedia sequence is ready to be launched after loading. Prior to sending this event, the Mode changed event will be sent. |
|  | Position changed | This event is generated continuously once every second to indicate the progress of playing a multimedia file. Use the mciGetPosition or mciGetPositionString functions to retrieve the current position.<br>See also page 472. |
|  | Size changed | This event is generated, when the MCI interface changes the size of the display window. |
|  | Media changed | This event is generated, when a new media (such as a CD) is loaded. |
|  | Error | This event is generated, when the MCI interface detects an error. The formula value of the MCI component contains the appropriate error string. |

➤ More information about the MCI component can be found on page 221.

## VBX

If you register a VBX, then it will usually augment the list of events by itself. Since these events are specific to the component, there is no way to document, here, which events will be registered. As is the case with all other component specific events, the events registered by the VBX will be appended to

the list. Some of the otherwise available events might be excluded from the list, since the VBX paradigm does not support them (such as drag&drop) between non-VBX components.

You should refer to the documentation accompanying the VBX in order to determine which events are/will be available.

## Encapsulation (*Client / Server*)

An encapsulation component will always register at least one event. In addition to this event, the developer of the encapsulation component (subassembly) will be responsible for exposing other facilities. These facilities, otherwise referred to as the sub-assembly's API (Application Programming Interface), will appear in the event list, as defined by the component developer:

| Icon | Event | Description (When generated) |
|---|---|---|
|  | Server has closed | This event is generated when the server has terminated normally. The client connected to this server will receive this event and can process it. |

## Database

The following events are generated by any installed database:

| Icon | Event | Description (When generated) |
|---|---|---|
|  | Record loaded | When a query is sent to a database, and a record has been returned to mindmap, an event of this type is generated. Also, if the database cursor has been moved to a new position, by means of the First record, Previous Record, Next record, Last record or Go to record commands. |

| | | |
|---|---|---|
|  | Delete failed | When the mindmap application attempts to delete a record in the database and the operation has failed, this event is generated. A possible cause for this failure might be that the database is write-protected. |
|  | Insert failed | When the mindmap application attempts to insert a record in the database and the operation has failed, this event is generated. A possible cause for this failure might be that the database is write-protected. |
|  | Update failed | When the mindmap application attempts to update a record in the database and the operation has failed, this event is generated. A possible cause for this failure might be that the database is write-protected. |

➤ More information about the database component can be found on page 146.

## Input/Output

The following events are generated, independent of the type of data stored on the clipboard.

| Icon | Event | Description (When generated) |
|---|---|---|
|  | New data in clipboard | Whenever something is copied to the clipboard, this event is generated. This event can be used to signal to the application that a copy has been successful. |
|  | Clipboard empty | Whenever another application has changed or has cleared the contents of the clipboard. |

➤ Read more about the input/output component and the clipboard on page 206.

## Menu

| Icon | Event | Description (When generated) |
|---|---|---|
|  | 'Menu' | When a menu component is placed on the screen and a text has been assigned to a field, then this text will appear as the event option in the event list. The text string itself has no meaning. It merely functions as a placeholder representing the $n^{th}$ menu entry.<br>Note: The only events a menu component has, are those entered into the component itself, by the developer. |

# Messages

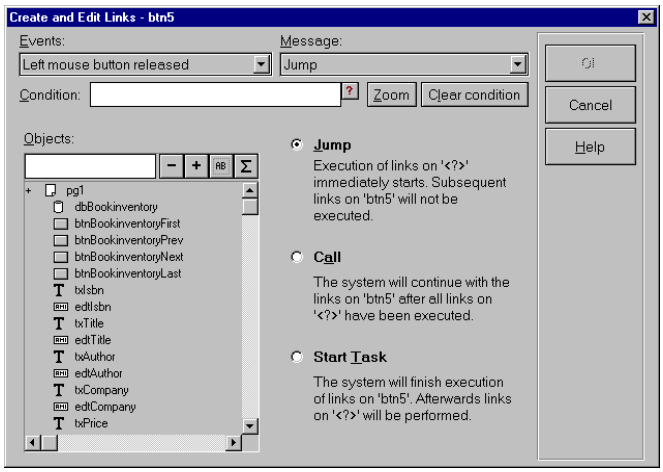
## Common Messages

The following list contains all the messages which are available in the standard environment. This means that even if you have not installed additional components, these messages will be available to you. Each additional component can choose to register its own messages. In this case, they are added to this list. The description of the component specific messages are dealt with on a component-by-component basis at the end of this list.

Whenever you select a message from the list on the right side of the dialog box, the lower section of the dialog box will change to reflect the selection. The actual contents of the link dialog box after having selected a message is, thus, a function of the selected message.

# Jump

Once you select the jump message, the dialog box will take on the following appearance.




This dialog lets you jump to a different component

This message is used in a number of different ways. Its most common use is to transfer control from one page to another. By no means is this its most important use, though. The fact is that a jump message transfers control to whatever it jumps to, making it suitable for a number of different situations, such as:

- reassigning focus,
- beginning a ‘subroutine’,
- a dummy operation to execute parser statements.

In order to actually jump to another component (which includes a page), merely select the component in the list which opens once you select this message. You can jump to any component, although the reaction due to the jump might differ.

If you jump to a page (not a component on the page), the jumped to page will display itself. If you have set any effects on the page, these will take affect. All components on the page will receive a message to paint themselves and to evaluate any

parser statement associated with them. The paint sequence corresponds to the order in which they were created, or more precisely the order in which they have been layered (foreground / background). They will be painted from background to foreground. The component farthest in background, capable of receiving focus, will receive it.

If you jump to a component on a different page, then the page will be displayed, the components will be painted in the order as described above, and any parser statements will be evaluated. The difference is that the default focus assignment is overridden by the component jump. If the component the jump is directed to is capable of receiving a focus, it and not the component farthest in background, will receive focus.

Special consideration must be given if the component that receives the jump has been set to be invisible. In this case the page will not be displayed. Let's look at an example. If you have placed a rectangle on page two, along with a number of other components and you jump from page one to the invisible rectangle, then all links associated with the rectangle (Goal of a jump), as well as the associated parser statements will be executed. You can use this feature to launch activities without letting the normal effects of a jump become visible.

If you jump to a component on the same page, more or less the same process takes place. The page is not redrawn though, unless some other link is executed that invokes a repaint. Again, if you jump to an invisible component, the jump and all links associated with the target component will be executed. If the component could receive focus, but has been set to be invisible, then it will not accept the focus and focus will remain where it originally was (again, assuming that no link is activated that otherwise changes focus).

A jump message is also used to initiate a subroutine. This is done by dropping a placeholder component (most commonly a rectangle is used) on the desktop. Next, all desired links are placed on the rectangle. The event they trigger on though is Goal of a jump. When the rectangle is jumped to, then it is actually receiving a Goal of a jump event. Lets view this process in more detail:

| Sending Component | Receiving Component | Description |
|---|---|---|
| Left mouse button released → Call to... | | The component transfers control to the receiving component. |
| | Left mouse button released → Change Attributes Color to Green | This link is not executed, since the control has been received not via user interaction on the component, but via transfer from another component. |
| | Goal of a jump → Sound | This link is processed, since control has been received through the jump message/event. The sound message is executed. |
| Next link' | | Control is back at the initial component. |

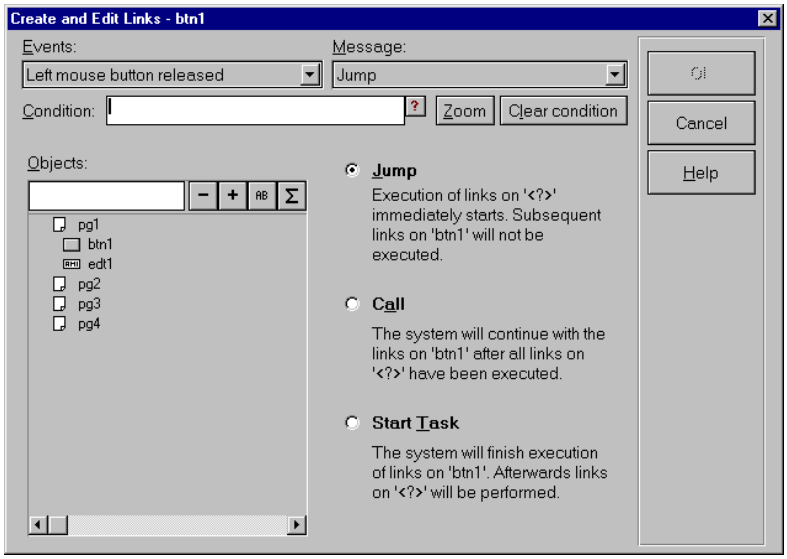
It must be noted, that the behavior of the sending component during the period in which the receiving component is processing its links, can be influenced by the setting of the switches on the right of the dialog box.

mindmap also supports computed jumps. When you select a component in the component list, mindmap will use the component as an absolute place to jump to. There might be situations in an application you are building, in which you would like to jump to a page or component which is determined at run time. To enable a computed (or parsed) jump, you must point the jump to a component which contains the destination. In this case the jump doesn't actually jump to the specified component, it uses the contents of the component as a pointer to the 'real' destination.

Let's use a little example to clarify this feature. Place an input field on Page1 of the application. Place a command button next to it. Now create a second and a third page. Go back to the first page and place a link on the command button which reads:

```
Left mouse button released   Jump to edt1
```

At the top of the section displaying the component list in the link dialog box, you can see a row of buttons:



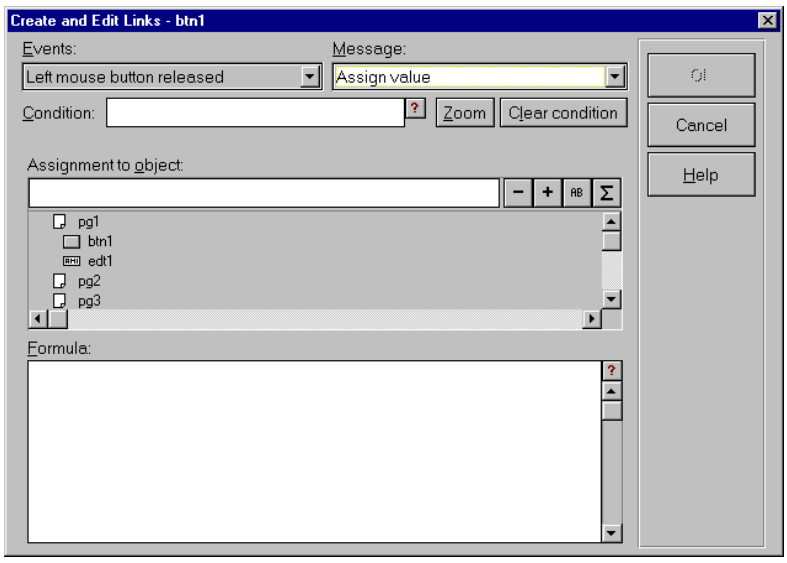

Chose a component as the target of the jump

Next to the plus (‘+’) and the minus signs (‘-‘) you will see two additional buttons. The button labeled ‘AB’ toggles the component list to display in alphabetical order. The button with the capital sigma (‘Σ’) is the one we need in this context. If you selected the input field labeled **edt1**, this name should appear in the area to the left of the four buttons. If you now click on the Σ-button, the list of components will be shaded and the component’s name listed in the field will be unshaded. The Σ-button will remained pressed. If you now leave the dialog box (by clicking the OK button) and return to the list of defined links, you will notice that the component’s name has been placed in square brackets (‘[ ]’). This symbolizes that the component, in this case **edt1**, is not being used as a jump target, but as a supplier of the pointer.

## Assign Value

Once you have selected this message, the dialog box will change to reflect the options associated with this

message:




This dialog assigns a computed value to a component

This is the method by which a specific value can be assigned to another component. If you employ the parser, then you can assign a statement to a component, which will be evaluated when the component is instructed to do so. The general conceptual syntax for the Assign value statement is

```
component x ← statement
```

with component x being the component selected in the upper section of the dialog box. (Please note that only those components that are capable of having a value assigned to them are displayed in the list). The statement belongs in the area beneath the list of components. The statement itself must adhere to a certain syntax.

➤ The valid syntax of statements is described extensively in conjunction with the parser on page 388.

A statement may include:

- constants or values
- operators

- references to other components
- functions

Thus, a valid statement might be:

```
sqrt(component x) * (10*pi)
```

or

```
substr(component y,2,3)
```

The first statement would assign the result of the mathematical operation on the component-x as described in the statement. The second statement would assign the result of the string operation to the appropriate component.

Please note that you do not include any assignment or equality operators such as '=' or '<' in the statement. By selecting the component in the list, you are already making the assignment and you must merely supply the rest of the 'equation'.

The component to which you assign the value, does not necessarily have to be able to deal with the assigned value. You can use a component as a storage location for values. Often, rectangles are used as temporary storage locations for values, much as they are used for subroutines.

Let's look at a little example of this concept:

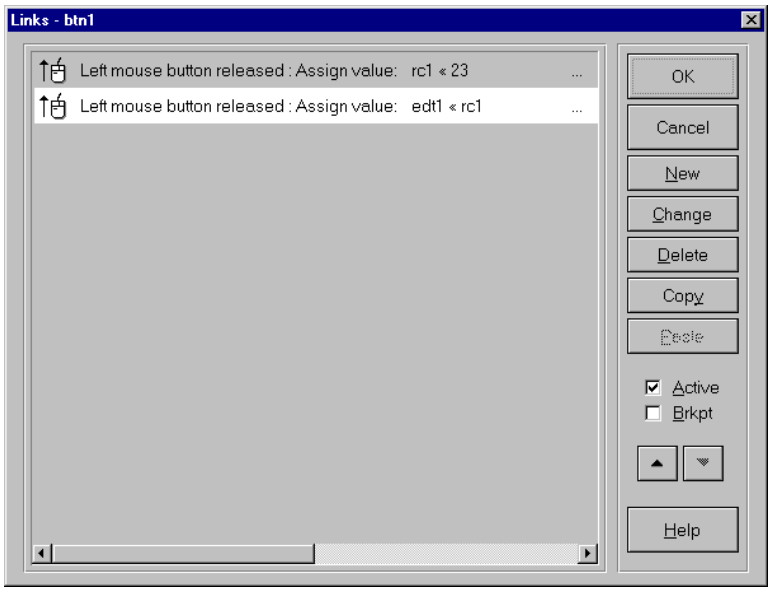
Place a command button, an input field and a rectangle, on the desktop. Select the command button and put the following link on it:

```
Left mouse button released- Assign Value 23 to rc1.
```

Next, define the following link on the same command button:

```
Left mouse button released- Assign Value rc1 to edt1.
```

Note that this assumes the rectangle has been given the name rc1 and the input field has been set to edt1.




Two links on btn1 assign values to different components

Please note that this little example would not function properly, if you were to reverse the order of the two links. The links are processed by mindmap in a top-to-bottom order.

The first link assigns the value to the rectangle and, even though the rectangle can't deal with the assignment in the conventional manner, it does store it. The second link takes the value of the rectangle and displays it, since the edit field knows how to deal with values.

## Drag&Drop

mindmap's mechanism of drag&drop is commonly used to move information from one component to another. While the assign value command is designed to evaluate formulas and store the result into a component's formula attribute, drag&drop can be thought of a mechanism which moves larger amounts of information, and especially types of information that cannot be calculated with, such as bitmaps.

Please also keep in mind that the drag&drop facility is used in conjunction with mindmap's encapsulation feature (client/server), which makes it possible to move information between components that reside on two different computers.

Drag&drop is accomplished using only a few rules. A component is either capable of:

- sending data,
- receiving data or
- both.

Information from a component can be offered in three different ways:

| Data Type | Description |
|---|---|
| ASCII | Text information, sometimes limited in size according to a component's limitations. Text can include multiple lines, separated by the carriage return-line feed sequence or multiple columns within the same line can be separated by the tab character. |
| BITMAP | Graphical information, usually device independent, i.e. rendering of an image occurs in the most suitable way with respect to the computer's graphic adapter. |
| METAFILE | Also graphical, but vector-oriented, data. |

All three data types are internally stored in the same format as used for the Windows clipboard.

Obviously, each component renders its information as a suitable type. Some components can even supply their data in multiple formats: a graphic component, for instance, can supply its contents as an image (either BITMAP or METAFILE), or it can send the image's file name, if requested by another component.

During the drag&drop exchange, both components negotiate a data type which both can support. Thinking of a drag&drop operation being established between graphic component and an input field, the graphic component will first offer its data as an image. Since an input field cannot deal with bitmaps, this offer will be refused, which in turn causes the graphic component to send the image's initial file name to the input field. If this negotiation also fails, nothing will be transferred, both components appear to be incompatible.

Not every component is capable of sending and/or receiving drag&drop information. The following list contains all components that participate in the drag&drop negotiation. Also, the data types that can be supplied or accepted are listed:

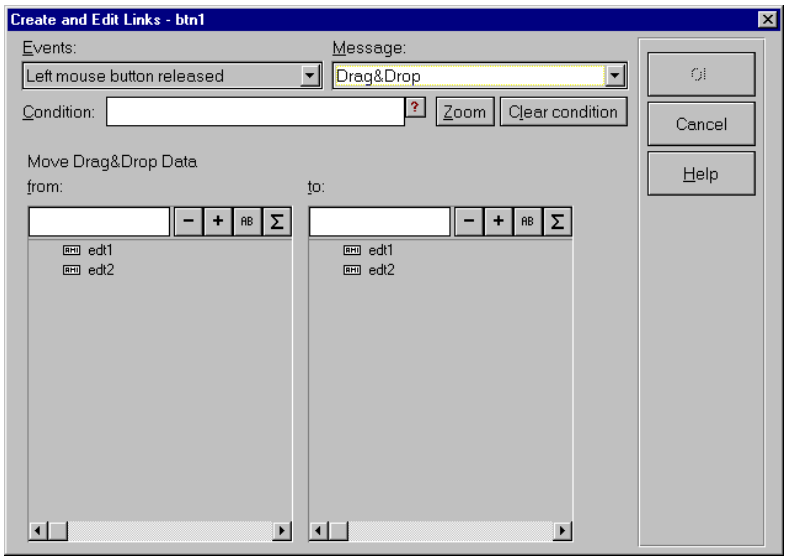
| Component | Sends | Accepts |
|---|---|---|
| Graphic component | Bitmap or MetaFile,<br>File name, if known, as text. The graphic component does not know about a file name if the image has been copied from the clipboard. | Bitmap or MetaFile.<br>Valid file name, i.e. complete directory, file name and extension information. According to the extension, the graphic component will choose a graphic input filter to load the image. |
| Input field | Text, limited by 32KB.<br>If the component is a multiline input field, the lines will be separated by the carriage return-line feed sequence.<br>Data classes other than string will be converted into strings before they are transferred.<br>An input field will always make it content available in the specified format (date, currency, etc.), as well as a string. | Text.<br>Large text data will be truncated to the limitation of the input field.<br>Should the input field be formatted to a class other than string, the information is expected to be a string and will be translated appropriately.<br>An input field will attempt to interpret the incoming data in its specified format (currency, date, etc.). If this fails, it will interpret it as a string. |
| List box | Text (file attribute not set):<br>Depending on the settings in the component's drag&drop settings, either the highlighted row(s) or all rows are sent. Rows are | Text (file attribute not set):<br>The received text is appended to the end of the list. If this text contains multiple lines, multiple rows will be created. |

| | | |
|---|---|---|
| | separated by the carriage return-line feed sequence.<br><br>Text (file attribute set):<br><br>The contents represents file names and will therefore be sent as such, including the full drive and path specification.<br><br>Graphic (Bitmap only):<br><br>If the list box is set to Show Bitmaps, the currently selected bitmap will be sent, along with the other representations of the data.<br><br>Collapsed list box:<br><br>The selected row will be stripped of its leading indentation marks ('\'), before it is sent. | Text (file attribute set):<br><br>The incoming text will be interpreted as a file name and displayed accordingly, including the full drive and path specification.<br><br>Graphic (Bitmap only):<br><br>If the list box is set to Show Bitmaps, a valid file name is expected. The list box will attempt to load the graphic file and stretch it to the width of the list box (keeping the ratio constant).<br><br>Collapsed list box:<br><br>The first characters will be interpreted as indentation markers. Each '\' will cause the following string to be indented by one column. |
| Data table | Text:<br><br>Multiple cells in the same row will be separated by the tab character. Multiple lines will be separated by the carriage return-line feed sequence. If a specific column has been selected in the drag&drop option dialog, only the contents of this column will be sent, separated by crlf.<br><br>The data table component either sends its entire content or only the highlighted rows, depending on the settings in the | Text:<br><br>To be copied to multiple cells of the data table, incoming text data is expected to contain multiple rows, separated by crlf and/or columns, separated by the tab character.<br><br>If multicolumn text information is received into a data table having selected a specific column in the drag&drop dialog will result in the data copied into subsequent columns beginning with the specified column. |

| | | |
|---|---|---|
| | drag&drop configuration dialog. | New rows will automatically be created if necessary. |
| Input/Output - clipboard | The component will send either text, bitmap or MetaFile data, depending on the current contents of the clipboard. If the clipboard is empty, nothing is sent. | The component will accept all types of information and copy it to the clipboard, making it available for other programs. |
| Input/Output - file | The component will send either text, bitmap or MetaFile data, depending on the contents of the selected file. If the file format cannot be determined, its contents will be sent as text. | The component will accept all types of information and create a file or append to an existing file if text data was received.<br>Make sure that a file extension suitable for the type of data has been specified.<br>If bitmap information has been received, the component will always create a device independent bitmap file (*.DIB or *.BMP). |
| Input/Output - printer | The component cannot send any information. | Text:<br>Text data will be printed on the specified printer using the printer's default font. The output is limited by a single page, regardless of the amount of data being received.<br>Graphic:<br>The printer will print the graphic. The size of the output can be modified using the component specific |

| | | |
|---|---|---|
| | | dialog. |
| Output page | The output page will send a rendering of the components on its page. | The component will either accept text or graphic and will create an appropriate component on its output page. |
| MCI component | The component cannot send any information. | Valid file name, i.e. complete directory, file name and extension information. According to the extension, the MCI component will attempt to open the media device. (same functionality as Open File link command. |

Once you select the drag&drop message, the dialog box will change its appearance. On the lower left-hand side of the dialog box, a list of components capable of supplying data via drag&drop is displayed. Immediately to the right, you will see a list of components in the application which have the ability to react to drag&drop data.




Select a sending and receiving component for the drag&drop mechanism

Let's use a simple example to explain how this message works. Place a button and two input fields on page 1. Select the command button and place a link on it. Use Left mouse button released as the event and select the Drag&Drop message. The two list boxes will only contain components capable of sending or receiving data via this message. Therefore, the button you have placed on the screen will not appear. Select edt1 as the component from which the data is requested and select edt2 as the receiving component. Put mindmap in run mode and type some data into the input field edt1. Now, click on the command button and the data should be copied from edt1 to edt2.

Let's take this one step further. Return to edit mode and click on the icon representing the import of graphic files. Select BMP as the format and search your system for a .BMP file. Generally, your Windows directory will contain such files. While you're at it, try to remember a second file name (or better yet, take a note of it). Accept the selection and place the bitmap on the desktop. Next, select the bitmap and set its attribute to drag&drop enable. (This is the icon with the little hand dropping the ball). You should still have a command button and an

input field placed on the desktop. If not, place one of each on the page. Select the command button and place a link on it. As an event, use the omnipresent Left mouse button released, and, as the message, use drag&drop. Now select the input field as the originating data source and select the bitmap as the receiving component. Accept the link, go back to the desktop and put mindmap into run mode. Your screen should look something like this:


A graphic component has received a file name through drag&drop. It loads an image file.

Select the input field and key in the name of the file you either memorized or have written down. Please make sure that you type in the complete path for the file. When you click on the command button, a drag&drop operation will be performed. Strangely enough though, the existing bitmap will be replaced by the bitmap contained in the file you have specified in the input field.

What has happened? The bitmap has received a file name as data via the drag&drop message. It has then attempted to display the file name, which it can't. It then attempted to locate a file by that name (and hopefully succeeded), has opened the file, read it, and displayed the contents in place of the existing

bitmap. The existing bitmap has taken on the roll of a placeholder.

Reverse the order of sending and receiving in the above example - drag&drop the bitmap to the input field (remember to set the attribute of the input field to enable incoming data via drag&drop). Go into run mode and execute the link. Interestingly enough, what will happen is that the bitmap will display its file name in the input field. What has happened now, is that the bitmap started out by sending the actual bitmap to the edit field. It responded by informing the sender, that it could not deal with this form of data, only with alphanumeric information. The bitmap component then sent it a file name. This the input field could deal with and immediately displayed it.

Try the same procedure with a list box. Place a list box on the desktop, along with a button and a bitmap. Set the drag&drop attribute of the list box to receive data. Instruct the button to perform a link, whereby it is to send data via drag&drop from the bitmap to the list box. Put mindmap into run mode and execute the link. As presumably expected, the list box displays the file name associated with the bitmap. Now go back into edit mode and set the specific attributes of the list box to 'Show Bitmaps' (bottom left hand corner of the specific attributes dialog box). Now get back into run mode and execute the link. Well, the list box now displays the bitmap you sent it. If you execute the link multiple times, the list box will display multiple instances of the bitmap.

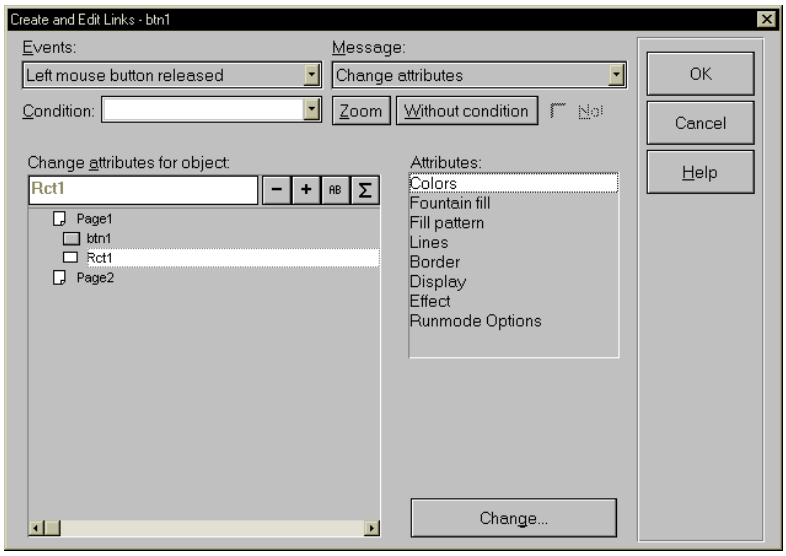
The drag&drop message also supports the computed drag&drop feature. When specifying the sending component, click on the Σ-button after selecting the component. Keep in mind though, that the supplying component must be capable of containing a pointer (i.e. a graphical primitive cannot be used here). Then select the destination. When the link is executed, it will go to the specified component, pick up the pointer, and get the contents of the component to which the pointer is pointing. The contents will then be forwarded to the receiving component.

The same procedure can be used the other way around. The receiving component can contain a pointer to the component which is really supposed to receive the data. This way you can pick up data from a component and send it right back to itself.

# Change Attributes

During the course of an application, it often becomes necessary to change one or many attributes of a component. You might wish to change the color of a graphical primitive to reflect a change in a certain status, or you might want to gray the text on a button and simultaneously switch it to be inactive.

To achieve this, you merely have to define the event which will trigger the change, select the component you want to perform the attribute change on, the attribute to be changed, and the new value it is to take on:




This link will modify an attribute of a component

All attributes that can be changed in edit mode, can also be changed in run mode. Obviously, this will always be at the discretion of the application developer, but, nonetheless, all attributes can be changed.

If a component registers its own attributes, these are accessible via the entry Object in the attribute list.

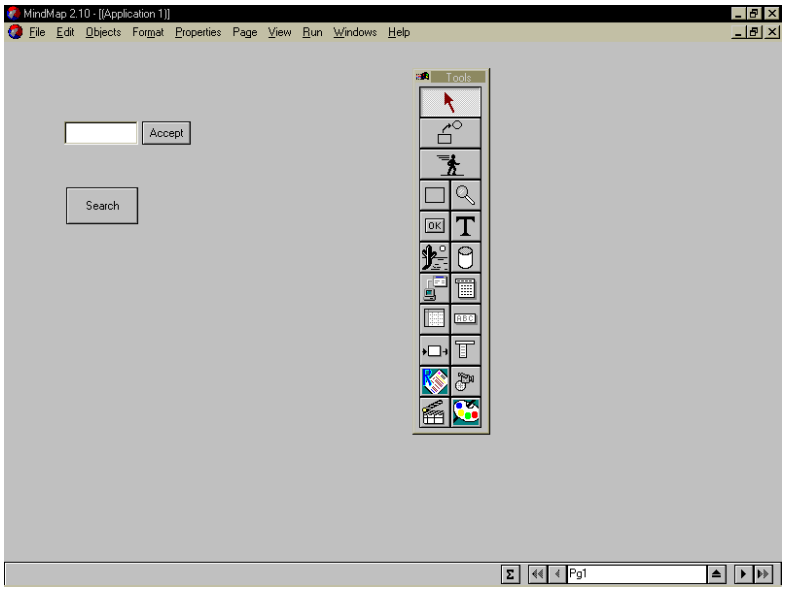
Let us use two examples to demonstrate how this message is used.

First, we will simply toggle the color of rectangle:

1. Place a command button and a rectangle.
2. Access the link facility on the command button.
3. The event should be Left mouse button released.
4. The message should be Change Attributes.
5. Select the rectangle in the list.
6. Double-click on the Colors attribute and select red.
7. Acknowledge the link.
8. Create a new link using the Right mouse button released as the event.
9. Proceed as described above, but select a different color this time.
10. Acknowledge the input and return back to the mindmap screen.
11. Put the application into run mode and toggle the color.

Now, let us create a somewhat more complex link, using the Change Attribute message.

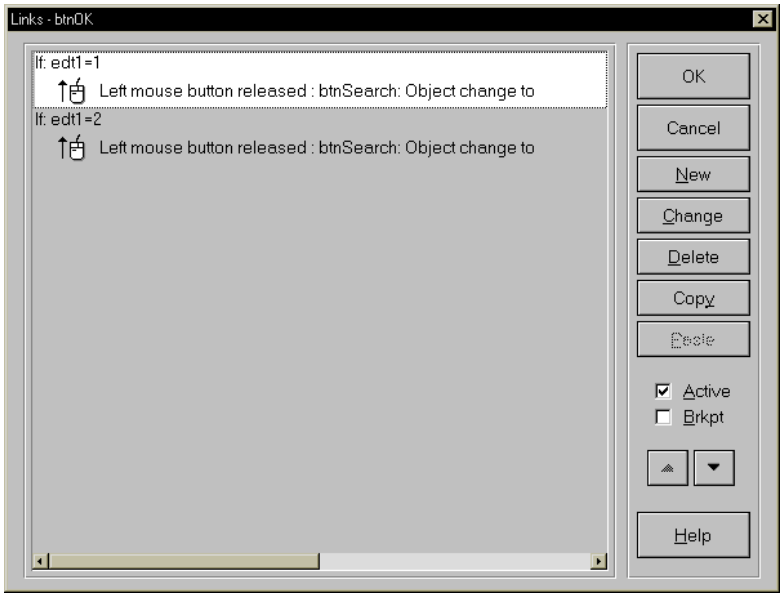
The goal is to turn a command button inactive, conditional on the contents of an input field:




A little application which changes an attribute of a component

1. Place two command buttons and an input field.
2. Label one command button with OK, label the other one with Search.
3. Name the first button, btnOK, the second one btnSearch.
4. Set the format of the input field to be Number.
5. Access the links on btnOK.
6. As the event use Left mouse button released.
7. As the message select Change Attributes.
8. Now select btnSearch and its Object attribute.
9. Check the check box labeled inactive.
10. Select the Conditional field just beneath the Event list and enter **edt1=1.**
11. Acknowledge the link.
12. Create a new link with the inactive check box unchecked and enter as a condition, **edt1=2.**

Your links should look like these:



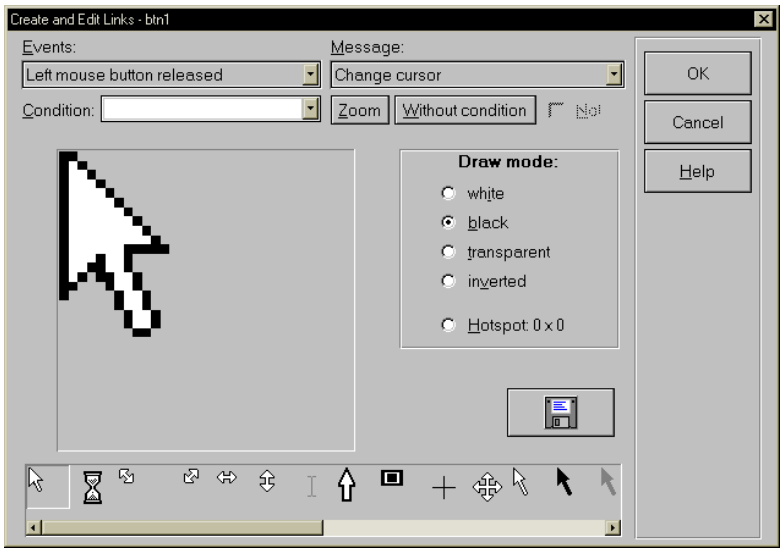

Two conditional links change the appearance of a button co m-ponent

Put this little application into run mode and enter a 1 into the input field. Once you have released the left mouse button on the Accept button, the Search button should become inactive. Entering a 2 into the input field, and subsequently releasing the left mouse button on the Accept button, will switch it back to the active mode.

## Change Cursor

This message is used when you want to notify the user of some, possibly non-obvious, condition in your application. Examples might be that you want to:

- cue the user as to the existence of a hot spot,
- signal that a special operation is taking place (printing, drag&drop, etc.)

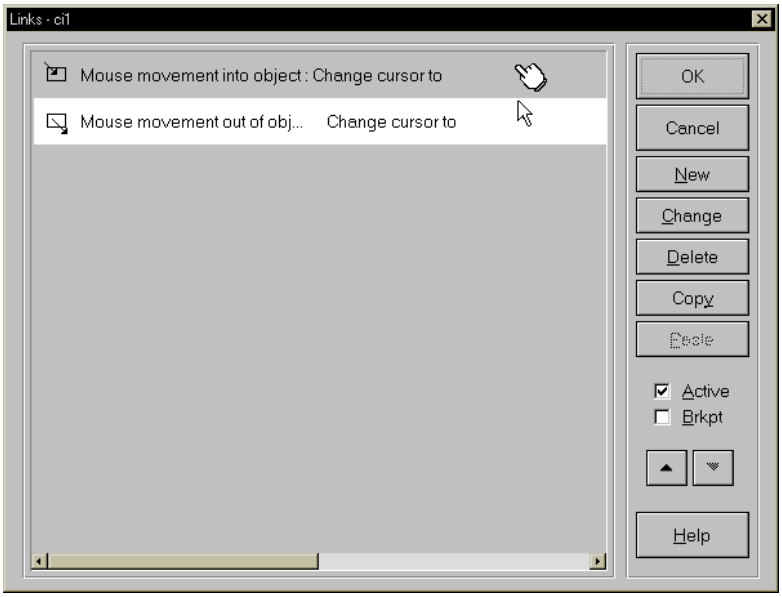



Select from several predefined mouse cursors or easily create your own

We will assume you want to change the cursor to indicate the location of a hot spot on a bitmap:

1. Place a bitmap on the screen.
2. Select a circle (graphical primitive) and place it on top of some significant area of the bitmap.
3. Select transparent as the fill color.
4. Select transparent as the line color.
5. Access the link facility on the circle.
6. Select Mouse Movement into Object as the event.
7. Select Change Cursor as the message.
8. Choose one of the cursors offered in the scrollable list.
9. Acknowledge the link.
10. Create a new link with Mouse Movement out of Object as the event and the same message.
11. Now, pick a different cursor from the list.
12. Acknowledge this link.

Your two links should look something like these:

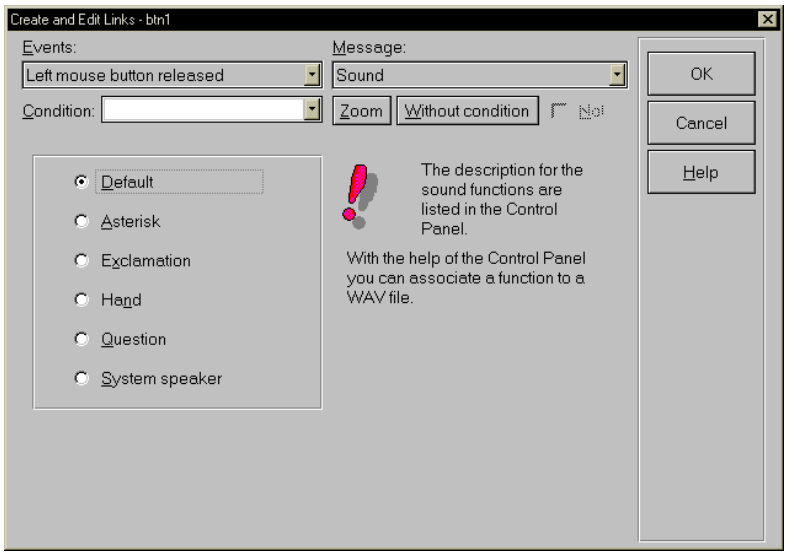



The mouse cursor will be changed when the mouse moves over a component

Keep in mind that you must always change the appearance of the cursor back to its initial state, dependent on some valid event, even if you jump to another page.

## Sound

This message is used to give the user some simple form of audible feedback, depending on some action. It will function on all Windows systems which have a loud-speaker. You should not confuse this message with the multi-media message MCI command and Multimedia. This message is the least common denominator on all Windows systems. Therefore, it is actually only capable of generating simple sounds, such as beeps.




Create different sound effects

This message picks up the valid settings on your system that have been assigned in the Windows control panel. Please note that your setting does not necessarily match those on other systems. Please keep this in mind when planning to deploy your applications to other systems.

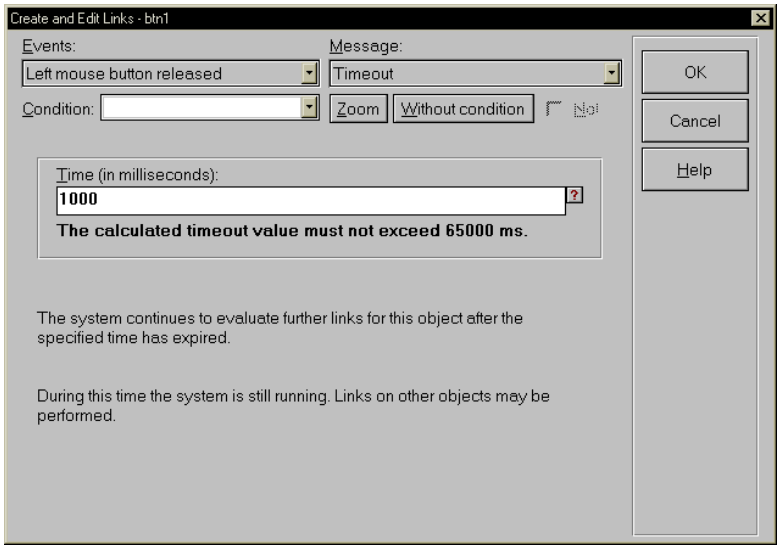
Building a link based on this message is extremely straight forward:

1. Place a component and access its links.
2. Select an event and pick the Sound message.
3. Click on one of the aforementioned sound options in the selection list.
4. Acknowledge the link and put the little application into run mode.
5. Generate the event and you should hear the sound.

## Time-Out

This message does exactly as the name implies — it times out for a specified period. During this period, all links on the component which starts the time-out are

not processed. Any other links on other components, which may become active during the time-out, will be processed.




This link pauses the execution of links for a given time

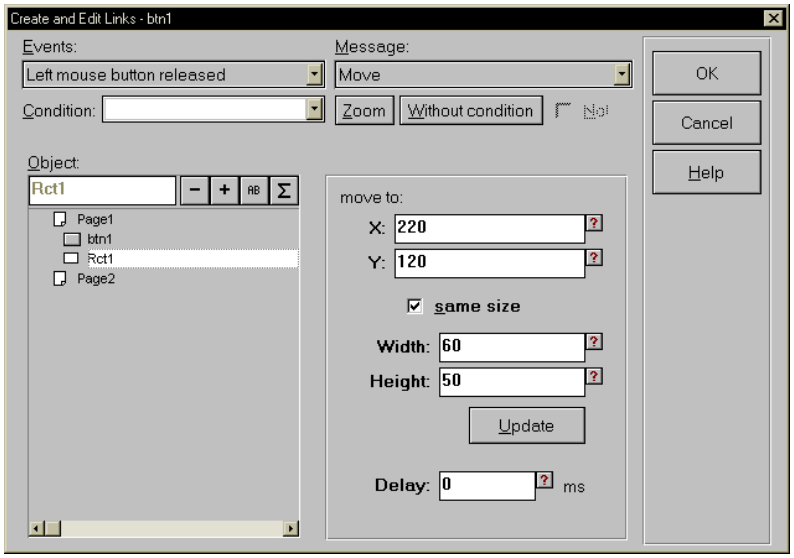
The time specified in the input field is defined in milliseconds (1 sec = 1000 msec). This field is also evaluated by the parser at run time, so that a statement can be input, instead of a constant. Thus, the actual time-out period can be kept variable.

## Move


This message allows certain components to be moved at run time.

The movement is limited to the page on which the component has been placed. Moving it outside the coordinate space of the page, will make it inaccessible for the user. Returning it into the visible area requires the execution of an additional Move message.

All components can be moved during run time, although in some cases, it is hard to conceive of a reason to do so.




This link moves a component in run mode

Again, let us build a simple example:

1. Place a command button and a rectangle.
2. Access the link feature on the command button.
3. As an event use Left mouse button released, as a message use Move.
4. Select the rectangle from the list of components below the event list.
5. Click on the Update button on the right side of the dialog box (This should cause the x- and y-coordinates of the rectangle to appear in the corresponding fields).
6. Retain the displayed Width and Height and enter a Delay of 1000 msec.
7. Acknowledge the entries and return to the mindmap screen.

Since all five fields are parsed at run time, you can also enter a computed statement.

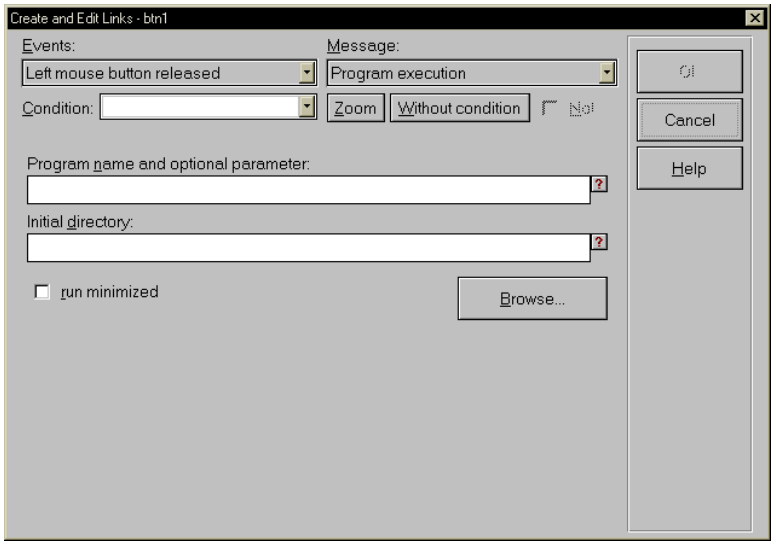
While still in edit mode, move the rectangle to a different location on the screen. Put the little sample application into run mode and click on the command button. With the delay of a second, the rectangle should return to the position at which it was prior to you moving it manually.

The Update button picks up the x- and y-coordinates, as well as the size of the selected component. Obviously, you can enter any other (valid) value. By default, the size of the selected component is retained.

When the component is moved with a delay, then a frame with the size of the component is drawn and it is moved with the specified delay to the designated screen position. It is not possible to move the component as it is seen in its static form.

## Program execution

This message is used to launch other executable files (or programs) from within mindmap. Keep in mind that launching an application in Windows implies that another process is started. This means that control will immediately return to mindmap -- links following the program execution will immediately be executed.




This link launches another application

Let's try a little example. To launch the MS Windows calculator from within mindmap, perform the following steps:

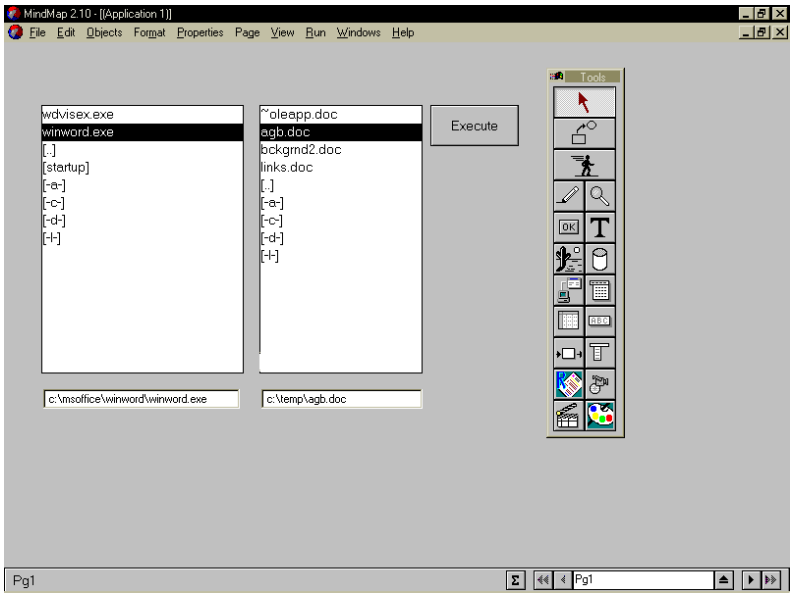
1. Place a command button.
2. Access the link feature on the command button.
3. Use Left mouse button released as the event and Program Execution as the message.
4. Click on the Browse button and navigate the file list until you find the calculator (under normal conditions, it will be located in your Windows directory).
5. Acknowledge the link and return to the mindmap screen.

You can use the parser function GetModuleHandle to verify if an application has already been loaded (see page 420 for a description of this function).

Place the little application into run mode and click on the command button. mindmap will pass control to Windows, which in turn will locate the Windows calculator and launch it. If you minimize the calculator, return to the little application and click on the command button again, another instance of the calculator will be launched. To avoid launching multiple instances, you must terminate the application you have started, before you attempt to launch it again.

Both fields in the dialog box are parsed, meaning you can also enter any valid statement. This is especially useful if you wish to launch the application in conjunction with a specific file. If you wish to launch a word processor, together with a document the user selected from a list, then you simply use the string concatenation feature to add the selected file to the program name.

Let's construct another little example. We want the user to pick the word processor application from one list of files and the document they want to work on from the second list. Clicking on the command button should launch the application and pass the document file, via the command line interface.




Here's an application that starts a program and automatically opens a document

Here are the necessary steps:

1. Place two list boxes. Name one ProgramFiles and the other DocFiles. Set the component specific attribute of both list boxes to display files with full path display.
2. Place an input field beneath each list box. Name the one beneath the ProgramFiles list box ProgramFileName. Name the other one DocFileName. (In a real application both input fields would be set to be invisible at run time).
3. Place a command button and label it with a caption, such as Execute.
4. Select the list box ProgramFiles and access the link facility. As an event, use Application Started. As the message, use Combo-/List box. Here, select the option, Directory, and enter the string "C:*.EXE", including the quotation marks. This will only display files with the extension .EXE.
5. Place another link on the list box. Use Double-click as the event and Drag&Drop as the message. Drag&drop the

contents of this list box to the input field ProgramFileName. (This will assure that the full path is used).

6. Do the same for the other list box, with the exception that the string you should enter is "C:*.DOC". This will display only those files with the extension .DOC.
7. Also place a link to drag&drop the contents of DocFiles to the input field named DocFileName.
8. Select the command button and access its link facility. Use Left mouse button released as the event and Program Execution as the message.

In some cases, it is necessary to load an application, but to keep it minimized. To accomplish this, mark the check box labeled *Run minimized*. If you launch an application using this switch, then the application will not appear in foreground, but it will be running.

9. Enter into the field labeled Program name... the following string
   ```
   ProgramFileName + " " + DocFileName
   ```
   The quotations marks enclose a blank which is concatenated between the program name and the doc file, since this is the convention by which files are specified in the command line (an old MS-DOS convention).
10. Return to the mindmap screen and place this little application into run mode. The left list box should fill itself with the top-level directories and the drives defined on your system. Navigate the structure until you find the program you wish to use. Every time you double-click on an entry in the list box, it will appear in the input field beneath the list box.
11. Navigate the right list box until you have located the document file you wish to work on. Here, again, every double-click will pass the selection from the list box to the input field.

Once you have located the program file and the document file, click on the command button.

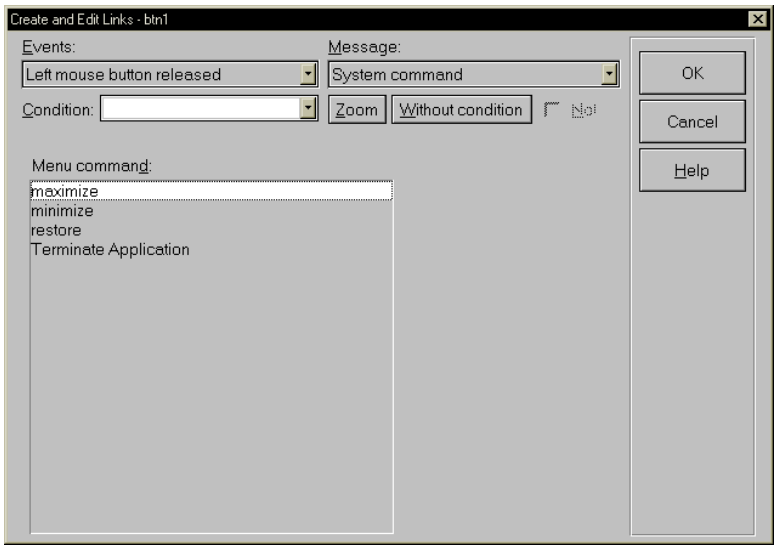
## System command

mindmap offers a set of commands which affect itself.

- Maximize: This option allows you to let the user expand the application to full screen.
- Minimize: Conversely, this option allows you to let the user minimize the application (iconize). If you have used the feature which allows you to define your own application

icons, this will be displayed, once the application has been minimized.

- Restore: Causes a mindmap application, that has been minimized, to restore itself to its initial size and position.
- Terminate Application: Executing this message will terminate the mindmap application and return control to the Windows environment. If the application you are executing is not running as an EXE file, then this message will function as a toggle back into the edit mode of mindmap.




A link can change the appearance of an application's window

## MCI Command

Windows offers a large set of multimedia extensions which can control audio/visual devices like sound boards, CD players, Video Disc players as well as the Video for Windows system.

All of these devices are controlled through a common interface. The basic approach to this multimedia control interface (MCI) is a command language. Windows itself defines only a minimum set of commands. Various types of multimedia devices

may introduce new commands to this syntax to cover their special features.

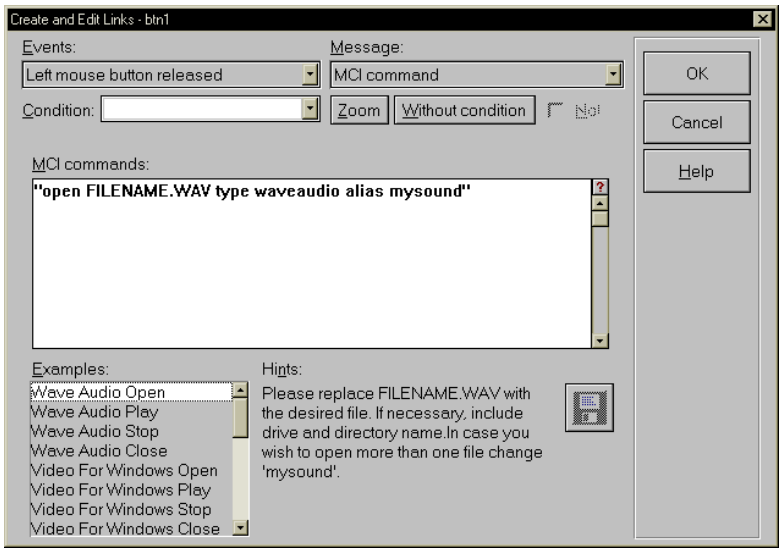
mindmap offers a convenient interface to this multimedia command language. The benefit of this approach is that low level calls to the device interface are possible. The disadvantage is that you will have to concentrate on the various options of a particular device by reading through the device's documentation.

The link dialog contains a generic set of predefined instructions you can chose from to build your MCI commands.

Please note at this point that a device is always identified by its type name. Valid type names may be:

| MCI Type | Description |
|---|---|
| waveaudio | Windows sound through a built-in sound board. |
| avivideo | Video for Windows if running under or if it has been properly installed as a Windows extension. |
| cdaudio | a CD-ROM drive as an audio CD-Player. |

Once an MCI device has been opened, it is referenced through an alias name in any further commands that apply to the open device. This device name is arbitrary. However, you should select a suitable name to make it easier to associate a particular statement with the type of device it is applied to.




Execute command strings on the MCI interface and control various MCI devices

If the device deals with files (Video for Windows and Wave, but not CD Audio), then the desired file name has to be included.

If you pick various open commands from the pick list at the bottom of the link dialog, you will find a FILENAME with the most common extension for this type of media. By pressing the button with the disk symbol, you may select an appropriate file from a standard file dialog box.

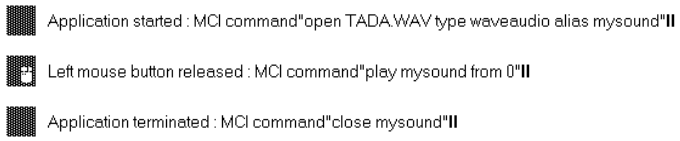
Since all MCI commands are parsed, you may freely construct your own MCI command strings from the contents of other mindmap components, such as input fields or list boxes.

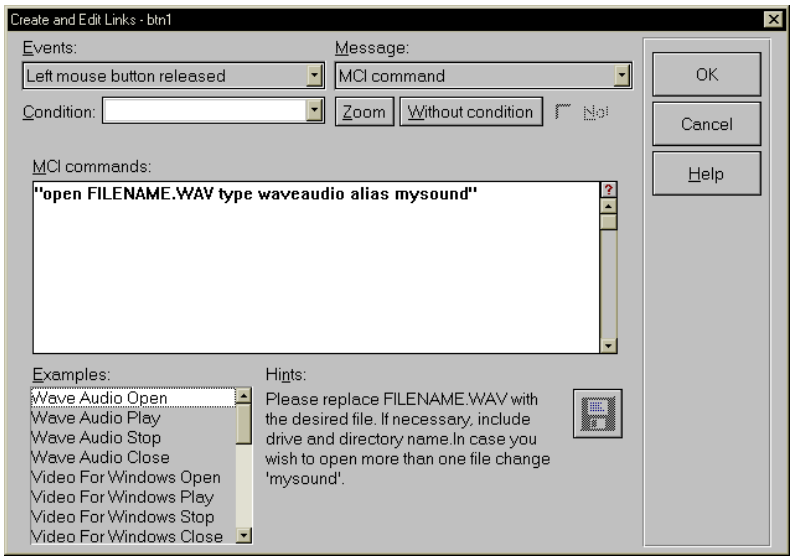
The following example will open and play a Windows wave file. Execution of subsequent links will pause, until the file has been played.

```
"open C:\\WINDOWS\\TADA.WAV type waveaudio alias
mysound"
"play mysound from 0 wait"
"close  mysound"
```

If you omit the word wait, mindmap will play the file and continue to execute subsequent links. In this case make sure that the sequence of commands is split across different events. Playing the sound file would immediately stop, if the MCI command close mysound would be executed immediately after the command play mysound from start.

Try to put the following sequence of MCI commands on a button, given that the wave file exists:



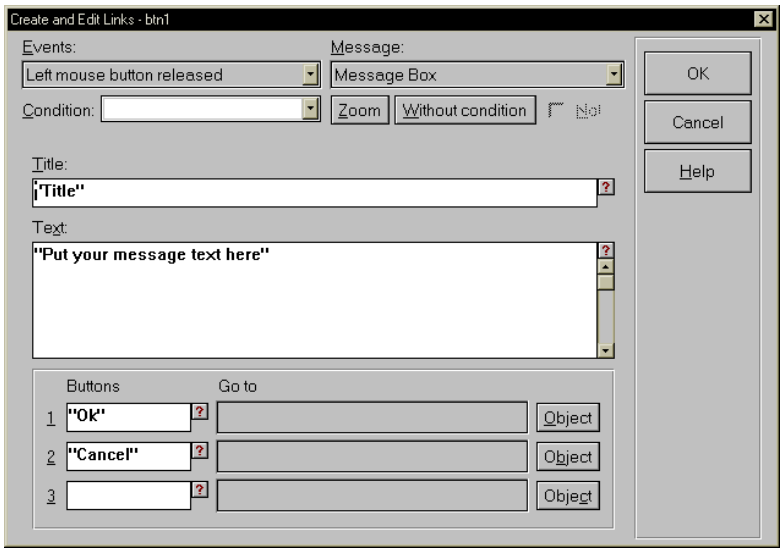

This link opens a sound file to be played on your sound card

➤ Another way to access MCI functionality is through the MCI component, described on page 221.

## Message Box

This message allows you to post a message box with various alternative buttons. This feature is generally used to inform the user of a certain condition, to offer options regarding the general flow of the application, or to post an error message.

Place a button on the mindmap screen and access its links. As an event, use Left mouse button released and select the Message box as the message. The following dialog box will be presented:




Fill out this form to prompt the user with a message box

The first entry you can make relates to the title or caption of the message box you wish to display. mindmap has put a placeholder into the field called Title. Please enter the desired caption into this field and make sure that you enclose the string in double quotation marks. Since this field is parsed, not enclosing a string in quotations marks will have the parser attempt to interpret the string as the name of a component which contains the desired string. The corollary is that you can also enter a component name containing the string you wish to use as a title.

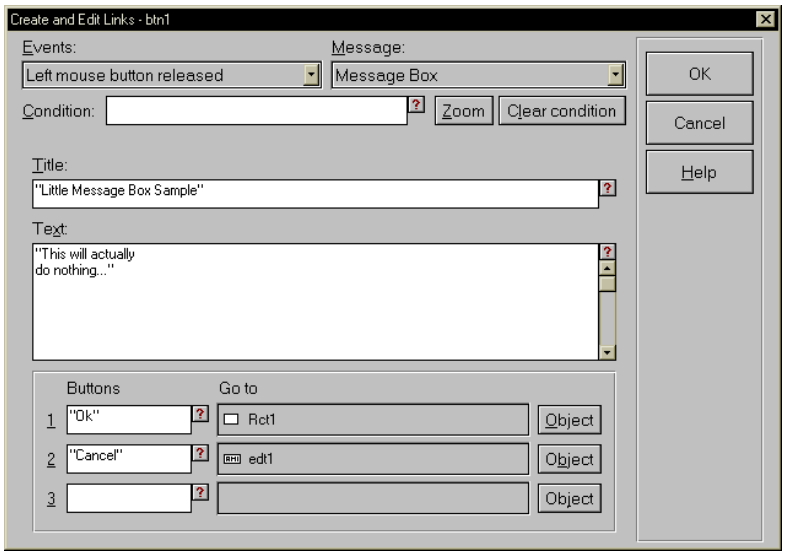
The next field is used to enter the actual body of the message text you wish to display. Again, this field is parsed. Thus, if you wish to enter a string which is to be displayed, then enclose it in double quotation marks. If, on the other hand, you wish to place a pointer (another component name) into this field, then do not enclose it in quotation marks. Please note that mindmap will automatically attempt to format the string, should it not fit into the boundaries of the message box. If you wish to force a new line (carriage return/line feed), then enter an **ENTER** or a **CTRL+ENTER**. In this case, the link will display a double vertical line, which depicts the position of the new line.

At the bottom of the dialog box you will find three fields which correspond to a maximum of three buttons you can display on the message box at run time. Enter the labels in the corresponding fields.

To the immediate right of each of the label fields, you will find a field titled Go To. This is where you can enter the location the button should jump to. You can either manually enter the component name to which you want to divert control, or you can click on the button labeled Object. This will pop up the list of all components.

Let's construct a little example. This example will display a message box with a caption and a body, along with two buttons pointing to other components:

1. Place a command button, an input field, and a rectangle.
2. Label the button.
3. Now access the link facility of the button, using Left mouse button released as the event and Message Box as the message.
4. Enter in a caption and a body for the message box.
5. Keep the OK label and point it at the rectangle (rc1).
6. Also retain the Cancel label and point it at the edit field.
7. Acknowledge the dialog and return to the desktop.




Inform the user about a certain condition of your program

Now let's deal with the components you are pointing to:

1. Select the rectangle and access its links.
2. As the event select Goal of a jump and as its message generate a Sound.
3. Acknowledge and return to the desktop.

Start your application. As soon as you click on the command button, the message box should appear. If you click on the OK button, the message box should disappear and you should hear a sound. Click on the command button again, this time click on the Cancel button. Again, the message box will disappear, but the focus will now be in the input field.

# Component Specific Messages

## Data Table

Since data tables are a standard mindmap component, this message will appear, even if you haven't placed a

data table in the application. You will not be able to execute a data table message though, until you have placed one in the application.

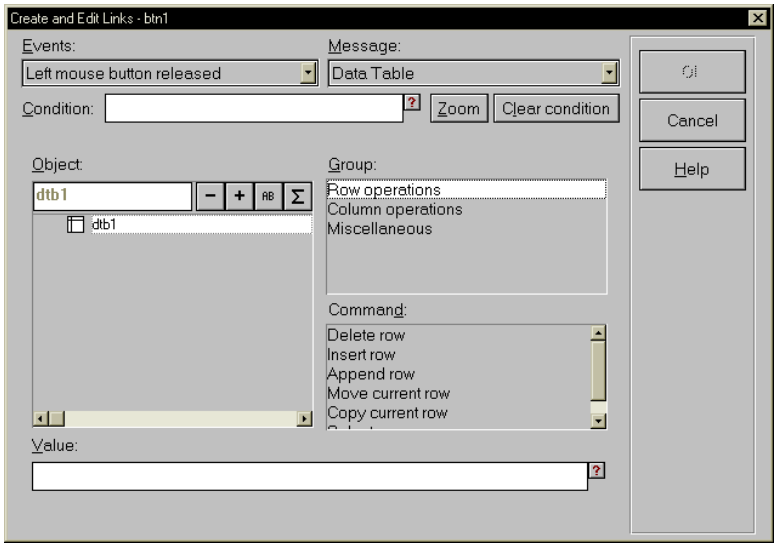
Once a data table is available, three classes of commands can be accessed. These are:

- Row operations
- Column operations
- Miscellaneous

As the labels suggest, the commands are either oriented at the row, the column or the table in general.

All row and column operations assume integer numbers. The data table message will automatically convert and/or round the values of strings, floating point, etc., to integers. Also note that row and column numbers start with 1.

Let's begin by looking at the row commands:




Select a function to change the appearance of a data table

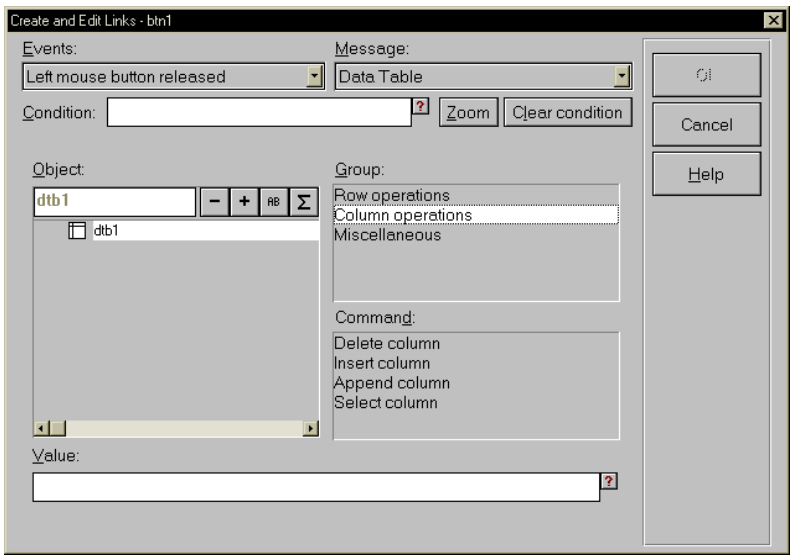
This group allows for six different commands:

| Command | Description |
|---|---|
| Delete row | This command will delete the row specified in the Value section of the dialog box. Since this field is parsed, you may either enter an integer or a component containing an integer. mindmap |

| | |
|---|---|
| | always attempts to resolve a string entry as a component name (reference), before interpreting it as a literal. Therefore, you can cascade the pointer in the Value field; i.e., a component name, which contains a component name, etc.<br><br>If the data table contains only one row, it cannot be deleted. |
| Insert row | This command will insert a new row immediately above the currently selected row. All rows - starting with and including the currently selected row - will be incremented by 1.<br><br>The contents of the newly inserted row will be empty.<br><br>The focus of the data table will remain with the previously selected row.<br><br>This command will override the setting of maximum rows in the component specific attribute. |
| Append row | This command will append a new row to the bottom of the data table.<br><br>The contents of the newly inserted row will be empty.<br><br>The focus will remain in the selected row.<br><br>This command will override the setting of maximum rows in the component specific attribute. |
| Move current row | This command will move the selected row to the new row specified in the Value field at the bottom of the dialog box. The moved row will be inserted into the new position. The old row from where the row was copied will be deleted.<br><br>Since this field is parsed, you may either enter an integer or a component containing an integer. mindmap always attempts to resolve an string entry as a component name (reference), before interpreting it as a literal. Therefore, you can cascade the pointer in the Value field; i.e., a component name, which contains a component name, etc.<br><br>If you do not specify a row number, the selected |

| | |
|---|---|
| | row will be deleted. |
| Copy current row | This command will copy the selected row to the new row specified in the Value field at the bottom of the dialog box. The copy of the row will be placed in a newly inserted row.<br><br>Since this field is parsed, you may either enter an integer or a component containing an integer. mindmap always attempts to resolve an string entry as a component name (reference), before interpreting it as a literal. Therefore you can cascade the pointer in the Value field, i.e. a component name, which contains a component name, etc.. |
| Select row | This command will select a row specified in the Value field at the bottom of the dialog box. Since this field is parsed, you may either enter an integer or a component containing an integer. mindmap always attempts to resolve an string entry as a component name (reference), before interpreting it as a literal. Therefore you can cascade the pointer in the Value field, i.e. a component name, which contains a component name, etc.<br><br>If you specify a row number less than 1 or greater than the largest existing row number, nothing will happen. Any previously selected row will remain selected. |

Now, let us review the column commands:




These functions deal with column manipulations of a data table

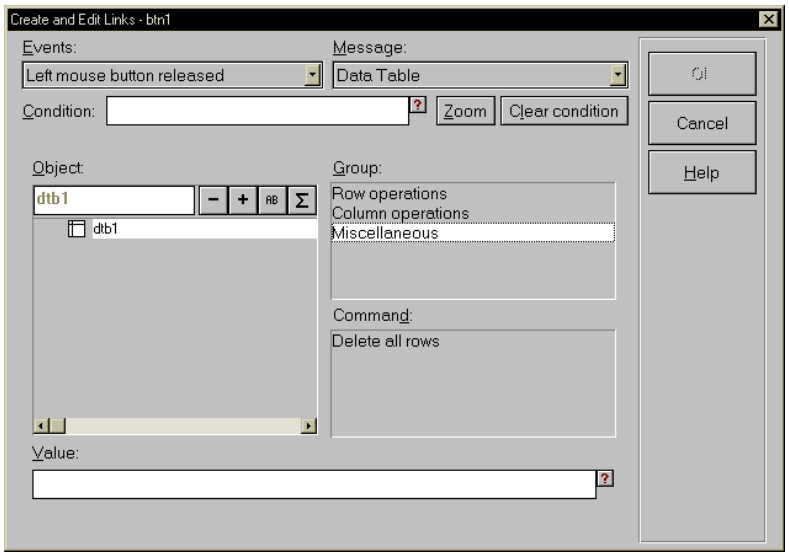
This group allows for four commands:

All column operations require that the switch '*Large Data*' be deactivated in the component specific attributes of the corresponding data table.

| Command | Description |
|---|---|
| Delete column | This command will delete the column specified in the Value section of the dialog box. Since this field is parsed, you may either enter an integer or a component containing an integer. mindmap always attempts to resolve a string entry as a component name (reference), before interpreting it as a literal. Therefore, you can cascade the pointer in the Value field; i.e., a component name, which contains a component name, etc.<br>If the data table contains only one column, it will be deleted. The consequence, though, is that the complete data table will also be deleted. |
| Insert column | This command will insert a column specified in the Value section of the dialog box. Since this field is parsed, you may either enter an integer |

| | |
|---|---|
| | or a component containing an integer. mindmap always attempts to resolve an string entry as a component name (reference), before interpreting it as a literal. Therefore, you can cascade the pointer in the Value field; i.e., a component name, which contains a component name, etc.<br><br>The insertion will increment the column number of all columns greater than the column being inserted by 1. The column attributes of the inserted column will be picked up from the settings in the component specific attributes of the data table.<br><br>Any column selections prior to the insertion will remain selected. |
| Append column | This command will append a new column to the right of the data table.<br><br>The contents of the newly inserted column will be empty.<br><br>The focus will remain in the selected row and column.<br><br>This command will override the setting of maximum columns in the component specific attribute. |
| Select column | This command will select the column specified in the Value section of the dialog box. Since this field is parsed, you may either enter an integer or a component containing an integer. mindmap always attempts to resolve an string entry as a component name (reference), before interpreting it as a literal. Therefore, you can cascade the pointer in the Value field; i.e., a component name, which contains a component name, etc.<br><br>mindmap will not highlight the currently selected column. |

Now, we will look at the last group:



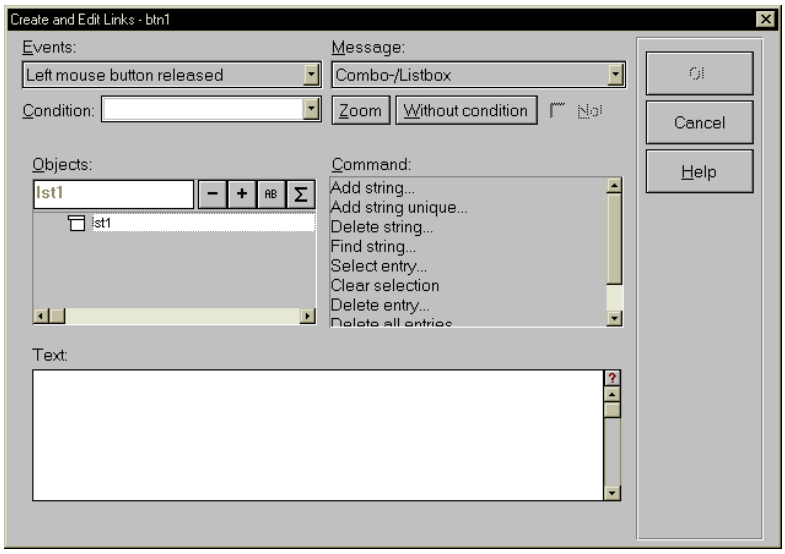

This function lets you completely erase the rows of a data table

This group contains one command, which deletes all rows. Column operations are still possible, such as insertion and deletion, but no rows exist. The display of the data table will only retain the column headings and the previous size.

- The Data table component is described on page 184.
- More information about events specific to this component can be found on page 248.

## Combo- / List boxes

This message is listed in the collection of standard messages, but it will only display its options if a combo-/list box has been placed in the application and selected in the left hand side of the dialog box. The available options are:




Depending on the configuration of a combo-/ list box, a pred e-fined selection of commands appears

The field labeled Formula at the bottom of the dialog box is parsed at run time. It will accept either literals enclosed in double quotation marks or standard parser statements.

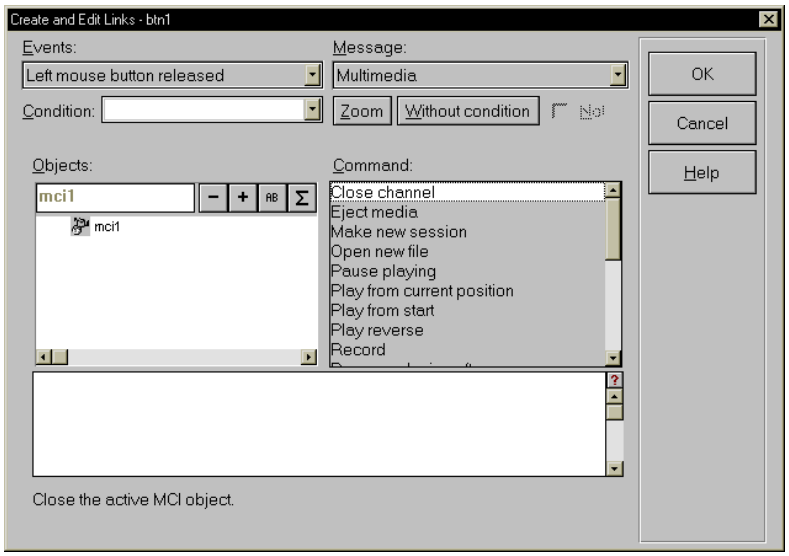
| Option | Description |
|---|---|
| Add string | This option will add the string referenced in the field at the bottom of the dialog box to the combo-/list box selected at the left hand side of the dialog box.<br>If the parser statement evaluates to multiple lines, e.g. by inserting crlf strings or by fetching the contents of a multi line input field, multiple insert operations will be performed. |
| Add string unique | This option will add the string referenced in the field at the bottom of the dialog box to the combo-/list box selected at the left hand side of the dialog box, only if the string is not already contained in the combo-/list box.<br>If the parser statement evaluates to multiple lines, e.g. by inserting crlf strings or by fetching |

| | |
|---|---|
| | the contents of a multiline input field, multiple insert operations will be performed if the lines are distinct. |
| Delete string | This option will delete an occurrence of the string which matches the string referenced in bottom of the dialog box. Regardless of the current position in the combo-/list box, the list will be scanned from the top of the list.<br>The comparison is case insensitive. |
| Find string | This option will find the first occurrence of the string which matches the string referenced in the bottom of the dialog box. Regardless of the current position in the combo-/list box, the list will be scanned from the top of the list.<br>The comparison is case insensitive. |
| Select entry | This option will select the row in the combo-/list box which is specified in the bottom of the dialog box. The option expects a number denoting a row number.<br>Row numbers are 1-based, meaning that the first row is number 1. Attempting to select a row number 0, will result in no row being selected. Attempting to select a row number greater than the number of existing rows will leave the last selected row still selected. |
| Clear selection | This option will remove the highlight of any selected row. If multiple selection has been defined for this component and multiple rows have been selected, then this operation will remove all selections. |
| Delete entry | This option will delete the row in the combo-/list box which is specified in the bottom of the dialog box. The option expects a number denoting a row number.<br>Row numbers are 1-based, meaning that the first row is number 1. Attempting to delete a non-existing row number will cause no row to be deleted. |
| Delete all entries | This option will delete all entries in the specified component, regardless of any selections. |
| Directory | This option will fill the specified combo-/list box |

| | |
|---|---|
| | with the contents of the directory referenced in the bottom of the dialog box.<br><br>The valid syntax for the specification is:<br><br>"x:*.*"<br><br>where x: is a valid drive; and *.* are wildcards for file names. You may also specify any other valid file name portion, such as *.BMP or XYZ*.*.<br><br>The entry must be enclosed in double quotation marks. Directory delimiters must always be two backslashes.<br><br>The field is parsed so that a component name or a valid parser statement can be substituted for a path specification in the form of a literal.<br><br>If the component specific attributes of the component were not set to allow the display of files, this message will automatically set the attribute, although it will only display file names (not drives and/or paths). In order for drives and/or paths to be displayed, the corresponding attribute settings must have been made.<br><br>If you have selected collapsed list box, then this operation will have no effect. |

## MCI

If you have multimedia capabilities installed on your system, then the MCI component will register a set of its own messages:




Yet another way to control multimedia devices - commands for a MCI component

The MCI component also registers a set of useful parser functions. See page 472.

| Command | Description |
|---|---|
| Open File | Open a media file. This command is applicable only to media that is associated with a file, e.g. Video for Windows or Wave Audio.<br>If the value input field at the bottom of the dialog contains or evaluates to a valid file name, then this file is opened. Otherwise a file open dialog box appears, which allows you to select a file. The system automatically loads a media driver that is suitable for the given file extension. |
| Close Channel | Closes a previously open media. You should use this command only if you control a device's behavior by means of special media commands. Usually, a device is automatically closed when its page becomes deactivated. |
| Play | Plays a previously opened media from the current position or from start, if it has not been played before. If the value input field contains a numeric value, playing starts from the given |

| | |
|---|---|
| | position. The units of this value depends on the type of media and its configuration (see the commands Use Frame Mode and Use Time Mode). |
| Play from Start | Plays a previously opened media from start. |
| Play Reverse | Play the media backwards, if applicable to the particular media. Note that Video for Windows files usually accomplish reverse playing showing single frames (step mode). |
| Stop | Stops playing the media. Playing resumes either through the Play or the Resume command. |
| Pause | Pauses playing of the media. The function is similar to the Stop command. However, if hardware media is directed by the media driver, this command differs from the stop command in that it sets the device in 'stand by' mode. For instance, a video disc player will keep the video heads running, while the Stop command will shut down the player. |
| Eject Media | Ejects the media, if applicable. Generally, CD audio drives and video disc players can eject the media. |
| Make New Session | This command creates a new media session. It is applicable only to media that has the ability to record. The value input field contains the name of the media device (e.g. 'waveaudio').<br>Note that Video for Windows movies generally cannot be recorded through the MCI interface. |
| Save File | Saves a previously recorded media stream to a disk file. The value input field contains a valid file name. Make sure that the extension of this file is suitable for the current media type. If this field is empty, a file save dialog box appears. |
| Use Frame Mode | Sets the media to use Frame Mode. (e.g. applicable for Video for Windows to return or set positioning information using video frames). |
| Use Time Mode | Sets the media to use Time Mode. (e.g. applicable for devices to return or set positioning information using the elapsed time from start). |

| | |
|---|---|
| Resume | Resumes playing of the device after a previous Pause command. |
| Seek To | Sets the position of the currently open media to the numerical value given in the input field at the bottom of the dialog. The units depend on the particular configuration of the device. (see the commands Use Frame Mode and Use Time Mode). |
| Send String | Can be used to send command strings to the media device (which must have been opened through either the Open File or the Make New Session command). This message allows full control over the device. Please refer to the documentation of the media specific driver. |
| Set Volume To | Sets the output volume to the level defined by the numeric value in the input field at the bottom of the dialog. Default volume is 1000 (= 100%). A value of 500 means half volume. |
| Set Speed To | Sets the playing speed to the level defined by the numeric value in the input field at the bottom of the dialog. Default speed is 1000 (= 100%). A value of 500 means half speed. |
| Set Repeat Mode | A value of 1 in the value input field indicates that the media has to be played repeatedly. 0 disables repeat mode. |
| Single Step | This command moves the current position of the media by a certain step relative to the current position. A value of 1 in the value input field means 1 step forward, -3 means three steps backwards. |

➤ For more information about the MCI component see page 221.

## Output Page

This component is the entity generally used to print information. Therefore, the specific messages it can deal with all relate in some manner to its printing.

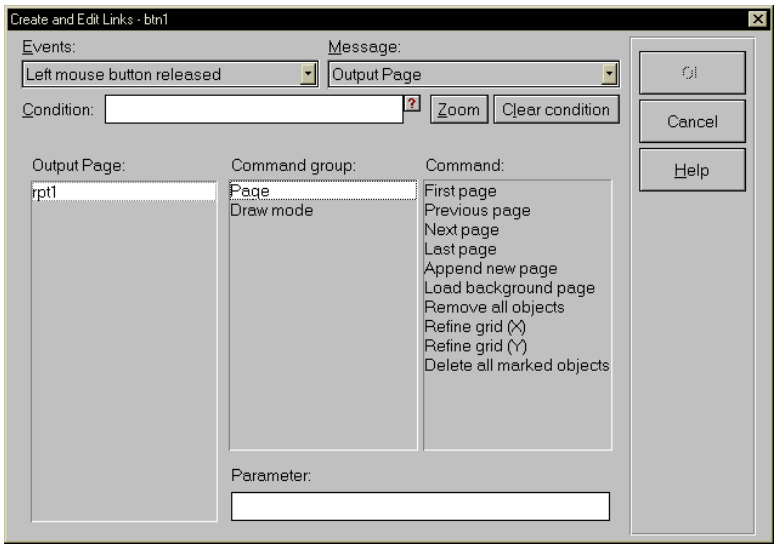
Since an output page is more or less another form of a mindmap page, components can be placed on it in the same manner as they are on mindmap pages.

Since this component is a standard component, the entry in the message list will appear, regardless of the placement of such a component in the application. The messages for the output page component are only available though, if an output page component has been placed in the application.

There are two groups of commands available for an output page component:

- Page
- Draw mode

We will first explore the Page group:



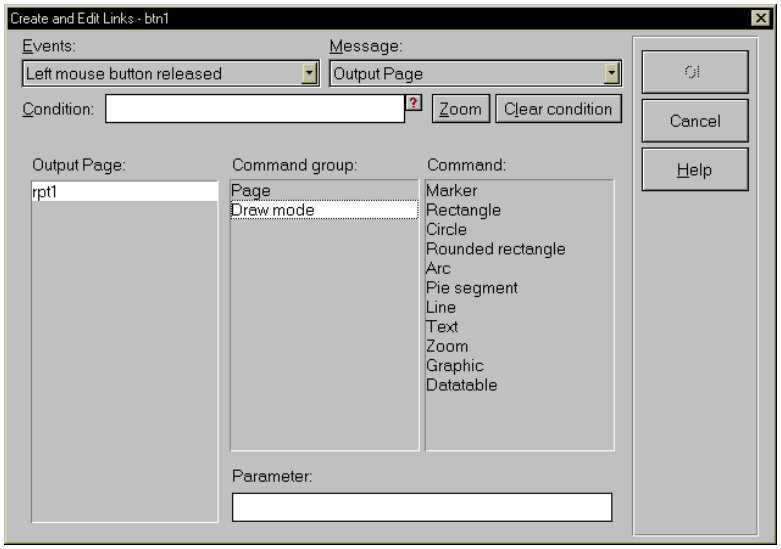

This dialog lets you manipulate the components on an output page

This group contains several commands:

| Command | Description |
|---|---|
| First page | If the output page component has more than one page, this command will position to the first page and display it. If only one page exists, this command will do nothing. |

| | |
|---|---|
| Previous page | If the output page component contains more than one page and it has not been positioned to the first page, then this command will position it to the previous page. |
| Next page | If the output page component contains more than one page and it has not been positioned to the last page, then this command will position it to the next page. |
| Last page | This command will position to the last page. If only one page exists, then this command will do nothing. |
| Append new page | This command will add a new page to the right of the last page and position to it. |

Next let's investigate the Draw mode group of commands:




A user may create new components on an output page at run time

Here we have the following commands:

| Command | Description |
|---|---|
| Marker | This command will allow the user to select components on the output page component. It is equivalent to the arrow on the toolbox when mindmap is in edit mode. |
| Rectangle | This command allows the user to draw a rectangle on the output page component at run time. The cursor remains in this draw mode, until a different function is selected. The standard procedure for accessing a component's attributes are also valid in run mode. |
| Circle | This command allows the user to draw a circle on the output page component at run time. The standard procedure for accessing a component's attributes are also valid in run mode. |
| Rounded rectangle | This command allows the user to draw a rounded rectangle on the output page component at run time. The cursor remains in this draw mode, until a different function is selected. The standard procedure for accessing a component's attributes are also valid in run mode. |
| Arc | This command allows the user to draw an arc on the output page component at run time. The cursor remains in this draw mode, until a different function is selected. The standard procedure for accessing a component's attributes are also valid in run mode. |
| Pie segment | This command allows the user to draw a pie segment on the output page component at run time. The cursor remains in this draw mode, until a different function is selected. The standard procedure for accessing a component's attributes are also valid in run mode. |
| Line | This command allows the user to draw a line on the output page component at run time. The cursor remains in this draw mode, until a different function is selected. The standard procedure for accessing a component's attributes are also valid in run mode. |
| Text | This command allows the user to draw a text component on the output page at run time. The cursor remains in this draw mode, until a |

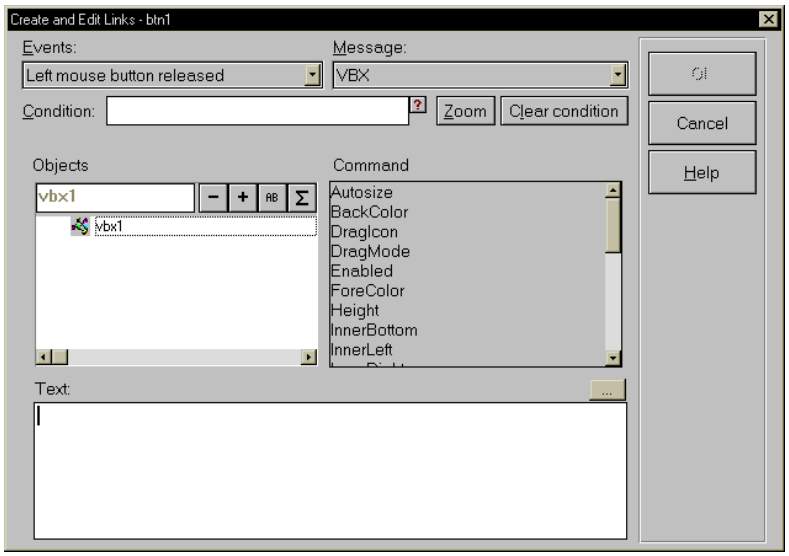
| | |
|---|---|
| | different function is selected. The standard procedure for accessing a component's attributes are also valid in run mode. |
| Zoom | This command displays the zoom toolbox, containing zoom in, zoom out and 1:1. The user can perform the zoom operations on the currently displayed output page component at run time. |
| Graphic | This command allows the user to import a graphic on the output page component at run time. The standard procedure for accessing a component's attributes are also valid in run mode. |

# VBX

This message is available, if you have included a VBX via the **File | Preferences** menu (see page 56). Only if the VBX has been placed in the application, will the actual messages be available, though.

The specific messages a VBX component is capable of dealing with is a function of the component itself. Each VBX has its own messages, so that you will have to consult the documentation accompanying the VBX.

The layout for the VBX message is consistent, independent of the selected VBX component. What differs are the actual commands, which are integral to the VBX component itself.




Every individual VBX component offers a specific list of commands to manipulate the appearance of the control

The field labeled Text accepts the parameters associated with the selected command. Thus it is also dependent on the VBX itself.

➤ For more information about the VBX component see page 226.

## Encapsulation (*Client/Server*)

This icon represents the messages associated with the encapsulation component relating to the server instance. It will only be available, if your application includes a server instance.

Once you select this message, mindmap will locate the server and display the server's default message and its API.

The default message which is available for all server instances is <<<Close the Server>>>.

This message instructs the application (subassembly) containing the server instance, to terminate normally.

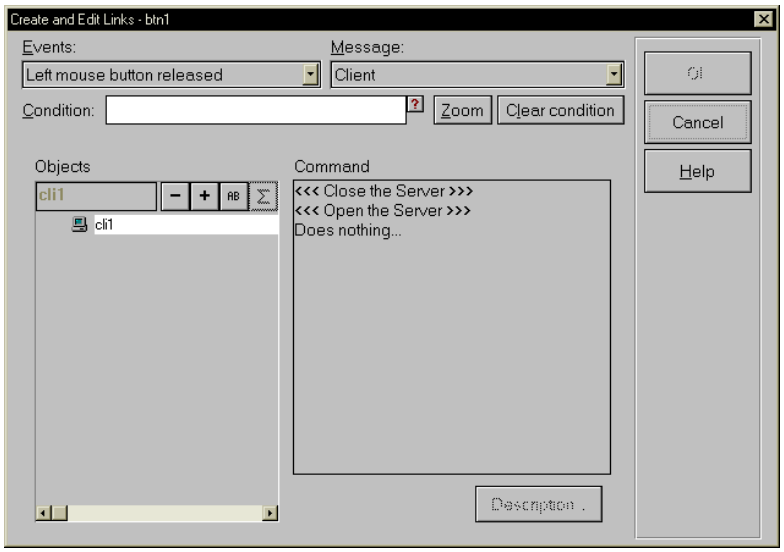
In addition to this default message, the server instance has those messages available that the builder of the instance de-

fined for it. These are the statements (API) that were defined in the process of constructing the server subassembly. Their function is dependent on whatever the designer intends them to do.

This icon represents the messages associated with the encapsulation component relating to the client instance. It will only be available, if your application includes a client instance.

Once you select the message from the list of available messages, mindmap will locate all client instances in your application and display them in the list. When you select one of these client instances, mindmap will locate the physical instance of the corresponding server (encapsulation) instance, open it, and query it for its API. If the associated server instance is not located on the same computer, mindmap will attempt to establish a physical connection to the computer on which the server component resides. It will then logically connect to the component and interrogate its interface. In this case, it might take some time to establish the connection, depending on the connection method (modem, ISDN, network, etc.). mindmap will display a progress bar during the process.

The result of this process is then displayed in the right hand section of the dialog box.




The link dialog for a client/ server component includes predefined commands as well as developer defined ones

The <<<Close the Server>>> message sends this command to the server instance, which causes it to close down. This message is automatically generated whenever the client component, which is connected to the server instance, is terminated normally. It is still suggested that you generate this message in a controlled manner from within your client application.

The <<<Open the Server>>> message must be sent to the server instance, before it will accept any other messages. Without receiving this message, the server instance will remain dormant.

Following these two default messages, are whatever other messages the constructor of the server application intended to expose to the outside world.

## Database

This message entry is visible even if a database component has not been placed in the current application. The actual message options are only available if a database has been placed.

In the case of the ODBC driver, the set of commands are identical, independent of the actual underlying database. In those cases in which the database is not a SQL compliant database, the ODBC driver emulates the actions of a true SQL database.

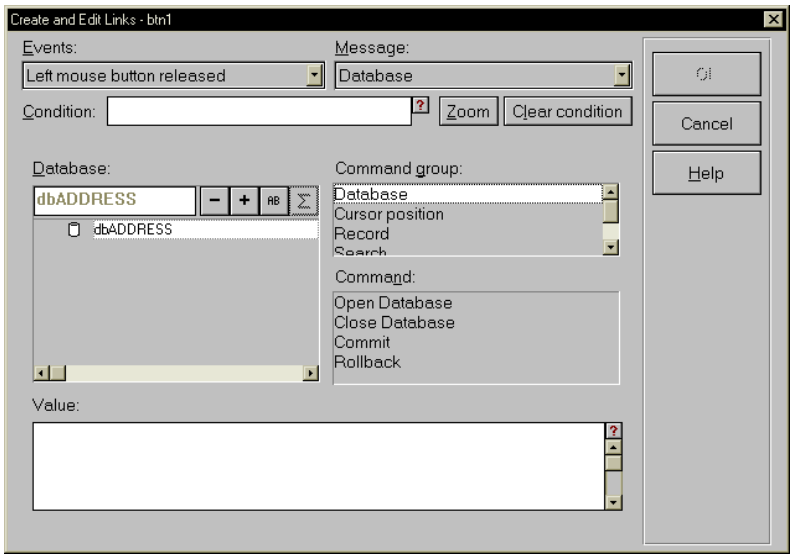
The database message contains five groups of commands. These are:

- Database
- Cursor position
- Record
- Search
- SQL Exec

Each one of these groups has a set of commands. The majority of the commands necessary to interact with a database have been summarized into high-level commands, making the input of SQL superfluous. These commands can usually be augmented by the inclusion of parameters specific to the command. In addition to these high-level commands, mindmap also offers the opportunity of explicitly entering SQL commands, thereby permitting complete control of the database.

Please note that it is beyond the scope of this manual to describe SQL (Structured Query Language) in detail. Despite a standardized group of commands and syntax, many database vendors have created specific extensions for their own databases. Again, it is beyond the scope of this manual to cover all of these variations. Please refer to vendor-specific manuals for more details.

We will begin by describing the first group:

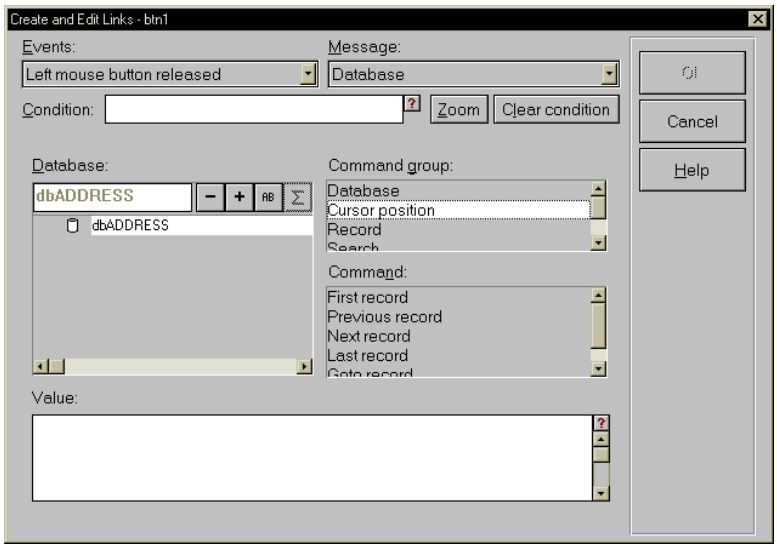



A database offers various commands for changing, searching, or navigating through a database

| Command | Description |
|---|---|
| Open Database | This command will open the selected database. If the driver accepts any additional options, these can be entered in the field at the bottom of the dialog box labeled Value. |
| Close Database | This command will close the selected database. |
| Commit | The default setting for a database is auto commit. If this attribute has been deselected, then the process of committing a database operation can be controlled manually. This command will have the database commit the operations. Please note that the particular ODBC database driver must support commit/rollback for this function to be available. |
| Rollback | The default setting for a database is auto commit. If this attribute has been deselected, then the process of rolling back a database operation can be controlled manually. This command will have the database rollback all |

| | |
|---|---|
| | non-committed operations. |

The second group of commands deals with the positioning of a cursor in the database. By definition, the database cursor can be moved to any record in the selected table. Operations such as deletion of records, insertion of records, etc., are then performed at the cursor position.



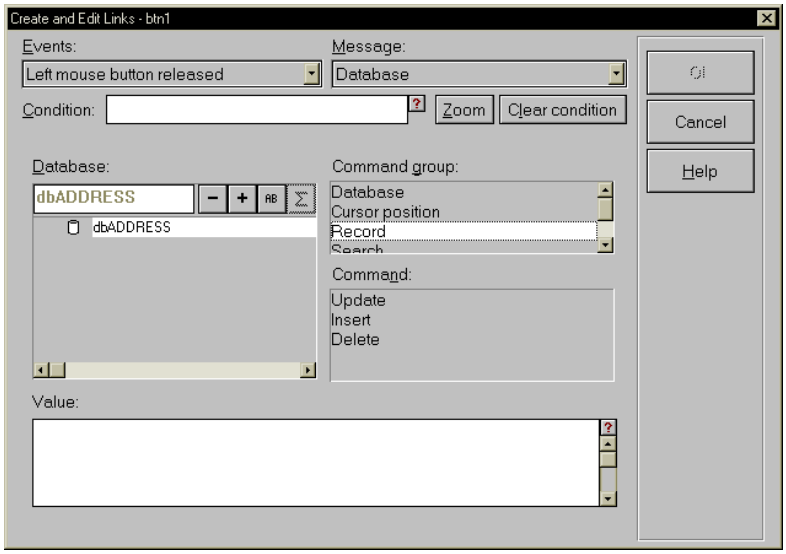

These commands let you browse through a database table

This group contains five commands. These are:

| Command | Description |
|---|---|
| First record | This command will position the cursor to the first record in the table. Record numbers are 1-based, so that the first record is number 1. |
| Previous record | This command will decrement the record counter by 1 and position the cursor at that point, if it is larger than 1. Otherwise, the cursor will remain at the first record. |
| Next record | This command will increment the record number by one and position the cursor at that |

| | |
|---|---|
| | point, unless the cursor is at the last record in the table, in which case nothing will happen. |
| Last record | This command will position the cursor to the last record in the table and set the record number to the last record. |
| Go To record | This operation will position the cursor to the record specified in the field labeled Value. The Value field will accept either an integer, or a component name which either contains an integer or another component name. The field is parsed at runtime.<br><br>Database record numbers are 1-based. |

The third group of commands deals with write operations to the database. These are:



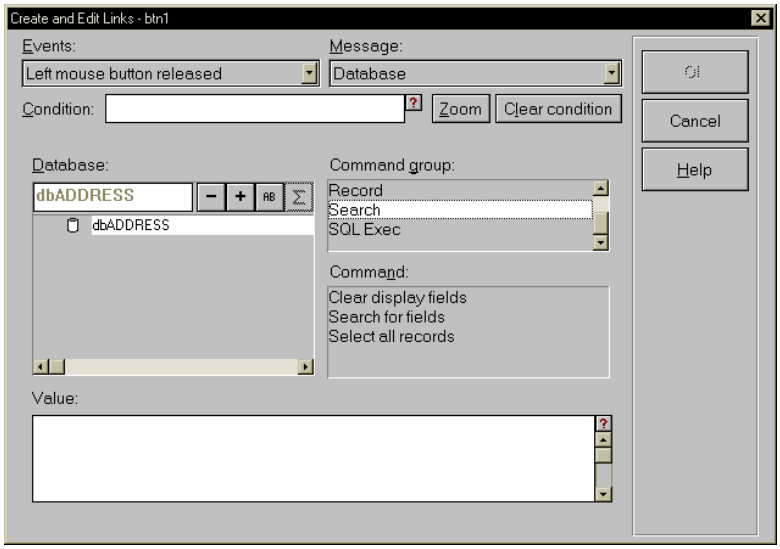

These commands are used to modify records in a database table

This group contains three commands:

| Command | Description |
|---|---|
| Update | This command updates an existing record and writes the contents of the components to the corresponding fields in the database table. The record to which the components are written is determined by the cursor position. The record at which the cursor is pointing at the time the operation is committed, is the record to which the data is written. Which components are written into their corresponding database fields depends on the particular assignment settings. See page 147 for more information.<br>No additional parameter (Value) is necessary. |
| Insert | This operation writes the contents of the components to the corresponding fields in the database. In doing so, this operation creates a new record in the database. Since relational databases are not kept in any specific order, the insertion occurs at the position of the cursor at the time the operation is committed. Which components are written into their corresponding database fields depends on the particular assignment settings. See page 147 for more information.<br>No additional parameter (Value) is necessary. |
| Delete | This operation deletes the record the cursor is pointing at, when the operation is committed.<br>Please be aware of the fact that deleting a row from a table may result in serious implications for the database.<br>Consider two tables containing customer information and employee information. An employee who is responsible for a particular customer may be identified in the customer table only by his ID. If you delete the employee, you will leave the corresponding ID in customers without reference to the employee table. This is a violation of referential integrity.<br>If not defined by the underlying database system, mindmap does not take special care |

| | |
|---|---|
| | about the referential integrity of the database and therefore will delete the selected record. |

The fourth group of commands deals with queries to the database. These are:



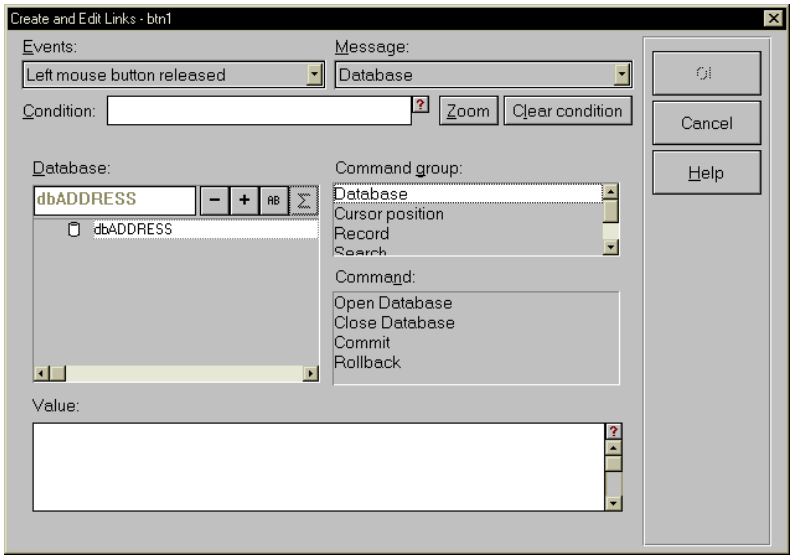

Use these commands to query the database

After search conditions have been entered in other mindmap components, use these commands to query the associated database table:

| Command | Description |
|---|---|
| Clear display fields | All input fields on any mindmap page associated with the selected database will be cleared. This means that their contents will be set to whatever has been previously specified in the corresponding default settings for each input field (Preset). Commonly, a null string has been set, so that the input fields will be blanked.<br>See page 193 for more information about input fields. |
| Search for fields | The default setting for an input field associated with a database is that it will be included in the |

| | |
|---|---|
| | query. Executing this command will construct a SQL SELECT statement which includes all fields that are set to be query fields. To exclude fields from participating in the query (being included in the SELECT statement), change the assignments in the connection between the input fields and the database table.<br><br>See page 184 for more information.<br><br>The exact structure for the SELECT statement is also determined by the search attribute of the corresponding input fields. |
| Select all records | This command will generate a SELECT * FROM table, resulting in a query which selects all records in the database. If dealing with a large database, be careful in executing this command, due to the possibility of an extremely large number of records being returned. |

The last group of commands allows you to expand the existing high-level database commands by means of explicitly generating SQL statements. For those users who wish to generate their own SQL statements, this is the location to do so. The commands included in this group are:




Enter your own SQL statements to gain full control over a database

| Command | Description |
|---|---|
| SQL SELECT | This command expects the WHERE, GROUP BY and/or HAVING clause of a SQL statement in the value editor at the bottom of this dialog.<br>mindmap will automatically prefix the given SQL part with the clause<br>`SELECT <connected fields> FROM<table name>`<br>Typically the given clause could be 'WHERE name like "Smi%". |
| SQL COMMAND | This message lets you enter and execute any type of SQL statement supported by the database. Use this command to typically issue commands which do not return a result set. |
| SQL COMMAND with refresh | This message lets you enter and execute any type of SQL statement supported by the database. You should use this command to execute SQL commands which return result sets.<br>Always make sure that the returned result set matches the assignment of mindmap components as defined by the database component. Retrieving data from the result set to undefined or invalid components may give unpredictable results. |

## Input/Output

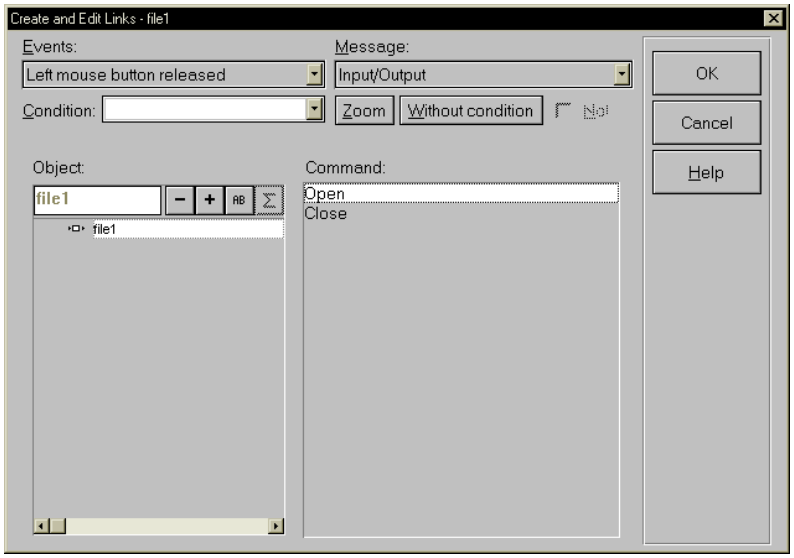
There are two different entities to/from which this command can be applied.

These are:

- File System
- Printer

Interaction with the file system implies that you either wish to read from or write to a file. Therefore, the appropriate messages are:

- Open
- Close




The input/ output component offers two messages to open or close a printer or file component

Note that printing in a network environment the input/output component will generate a new print job, whenever a printer is opened and closed. It is more common to put multiple print operations into a single job. Use these two commands to open and close the print job accordingly.

Select the input/output component from the list on the left side of the dialog box. Then select the message, depending on whether you are reading or writing a file. If you wish to read from a file, then drag&drop the contents of this component to a component which can deal with the type of data you are reading (i.e., input field for alphanumeric data, a bitmap for incoming or outgoing graphic data, etc.). Refer to the attributes of the component to set such things as file name, prompt user for file name, etc.

These two commands cause the file or printer being opened and closed through a link message. Usually, a drag&drop operation, that acts on either the printer or a file, will automatically open and then close the appropriate channel. Issuing multiple drag&drop commands after another, will cause unwanted open and close actions on the printer or file. Therefore, it is more convenient to open the appropriate channel first and close it when no longer needed.