Часто программисту приходится сталкиваться с задачей написания приложения, работающего в фоновом режиме и не нуждающегося в месте на Панели задач. Если вы посмотрите на правый нижний угол рабочего стола Windows, то наверняка найдете там приложения, для которых эта проблема решена: часы, переключатель раскладок клавиатуры, регулятор громкости и т. п.

Ясно, что, как бы вы не увеличивали и не уменьшали формы своего приложения, попасть туда обычным путем не удастся. Способ для этого предоставляет *Shell API*.

Те картинки, которые находятся в **System Tray** — это действительно просто картинки, а не свернутые окна. Они управляются и располагаются панелью **System Tray**. Она же берет на себя еще две функции: показ подсказки для каждого из значков и оповещение приложения, создавшего значок, обо всех перемещениях мыши над ним.

Весь **API System Tray** состоит из 1 (одной) функции:

```
function Shell_NotifyIcon(dwMessage: DWORD; lpData: PNotifyIconData): BOOL;
PNotifyIconData = TNotifyIconData; TNotifyIconData = record cbSize: DWORD; Wnd: HWND; uID: UINT; uFlags: UINT; uCallbackMessage: UINT; hIcon: HICON; szTip: array [0..63] of AnsiChar; end;
```

Параметр **dwMessage** определяет одну из операций: **NIM_ADD** означает добавление значка в область, **NIM_DELETE** — удаление, **NIM_MODIFY** — изменение.

Ход операции зависит от того, какие поля структуры **TNotifyiconData** будут заполнены.

Обязательным для заполнения является поле **cbsize** — там содержится размер структуры. Поле wnd должно содержать дескриптор окна, которое будет оповещаться о событиях, связанных со значком. Идентификатор сообщения Windows, которое вы

хотите получать от системы о перемещениях мыши над значком, запишите в поле **uCallbackMessage**
.

Если вы хотите, чтобы при этих перемещениях над вашим значком показывалась подсказка, то задайте ее текст в поле **szTip**. В поле **UID** задается номер значка — каждое приложение может поместить на
System Tray
сколько угодно значков. Дальнейшие операции вы будете производить, задавая этот номер.

Дескриптор помещаемого значка должен быть задан в поле **hIcon**. Здесь вы можете задать значок, связанный с вашим приложением, или загрузить свой — из ресурсов.

Примечание

Изменить главный значок приложения можно в диалоговом окне *Project/ Options* на странице
Application
. Он будет доступен через свойство Application.Icon. Тут же можно отредактировать и строку для подсказки — свойство Application.Title.

Наконец, в поле **uFlags** вы должны сообщить системе, что именно вы от нее хотите, или, другими словами, какие из полей **hicon, uCaiibackMessage** и **szT ip**
вы на самом деле заполнили. В этом поле предусмотрена комбинация трех флагов:
NIF_ICON, NIF_MESSAGE
и
NIF_TIP
. Вы можете заполнить, скажем, поле
szTip
, но если вы при этом не установили флаг
NIF_TIP
, созданный вами значок не будет иметь строки с подсказкой.

Два приведенных ниже метода иллюстрируют сказанное. Первый из них создает значок на System Tray, а второй — уничтожает его.

const WM_MYTRAYNOTIFY = WMJJSER + 123; procedure TForml.CreateTraylcon(n:Integer); var nidata : TNotifyiconData; begin with nidata do begin cbSize := SizeOf{TNotifyiconData) ; Wnd := Self.Handle; uID := n; uFiags := NIF_ICON or NIF_MESSAGE or NIFJTIP; uCallBackMessage := WM_MYTRAYNOTIFY; hicon := Application.Icon.Handle; szTip := 'THis is Traylcon Example'; end; Shell_NotifyIcon(NIM_ADD, @nidata); end; procedure TForml.DeleteTraylcon(n:Integer); var nidata : TNotifylconData; begin with nidata do begin cbSize := SizeOf(TNotifylconData); Wnd := Self.Handle; uID := n; end; Shell_NotifyIcon(NIM_DELETE, @nidata); end; end; **Примечание**

Не забывайте уничтожать созданные вами значки на System Tray. Это не делается автоматически даже при закрытии приложения. Значок будет удален только после перезагрузки системы.

Внешний вид значка, помещенного нами на System Tray, ничем не отличается от значков других приложений (рис. 1).

Рис. 1 Над значком, помещенным на панель System Tray, видна строка подсказки.

Сообщение, задаваемое в поле **uCallbackMessage**, по сути дела является единственной ниточкой, связывающей вас со значком после его создания. Оно объединяет в себе несколько сообщений. Когда к вам пришло такое сообщение (в примере, рассмотренном выше, оно имеет идентификатор **WM_MYTRAYNOTIFY**), поля в переданной в обработчик структуре типа **TMessage** распределены так.

Параметр **wParam** содержит номер значка (тот самый, что задавался в поле **uID** при его создании), а параметр **LParam** — идентификатор сообщения от мыши, вроде **WM_MOUSEMOVE, WM_LBUTTONDOWN** и т. п. К сожалению, остальная информация из этих сообщений теряется.

Координаты мыши в момент события придется узнать, вызвав функцию **API GetCursorPos** :

```
procedure TForml.WMICON(var msg: TMessage); var P : TPoint; begin case msg.LParam of WM_LBUTTONDOWN: begin GetCursorPos(p); SetForegroundWindow(Application.MainForm.Handle); PopupMenul.Popup(P.X, P.Y); end; WM_LBUTTONUP : end; end;
```

Обратите внимание, что при показе всплывающего меню недостаточно просто вызвать метод **Popup**. При этом нужно вынести главную форму приложения на передний план, в противном случае она не получит сообщений от меню.

Теперь решим еще две задачи. Во-первых, как сделать, чтобы приложение минимизировалось не на Панель задач (TaskBar), а на System Tray? И более того — как сразу запустить его в минимизированном виде, а показывать главную форму только по наступлении определенного события (приходу почты, наступлению определенного времени и т. п.).

Ответ на первый вопрос очевиден. Если минимизировать не только окно главной формы приложения Application.MainForm.Handle, но и окно приложения Application.Handle, то приложение полностью исчезнет "с экранов радаров". В этот самый момент нужно создать значок на панели System Tray. В его всплывающем меню должен быть пункт, при выборе которого оба окна восстанавливаются, а значок удаляется.

Чтобы приложение запустилось сразу в минимизированном виде и без главной формы, следует к вышесказанному добавить установку свойства Application.showMainForm в значение *False*. Здесь возникает одна сложность — если главная форма создавалась в невидимом состоянии, ее компоненты будут также созданы невидимыми.

Поэтому при первом ее показе установим их свойство **Visible** в значение *True*. Чтобы не повторять это дважды, установим флаг — глобальную переменную **shownonce** :

```
procedure TForml.HideMainForm; begin Appiication.showMainForm := False; ShowWindow(Application.Handle, SW_HIDE); ShowWindow(Application.MainForm.Handle, SW_HIDE); end; procedure TForml.RestoreMainForm; var i,j : Integer; begin Appiication.showMainForm := True; ShowWindow(Application.Handle, SW_RESTORE);
```

```
ShowWindow(Application.MainForm.Handle, SW_RESTORE);     if not ShownOnce then
begin       for I := 0 to Application.MainForm.ComponentCount -1 do     if
Application.MainForm.Components[I] is TWinControl then       with
Application.MainForm.Components[I] as TWinControl do      if Visible then              begin
 ShowWindow(Handle, SW_SHOWDEFAULT);              for J := 0 to ComponentCount -1 do
 if Components[J] is TWinControl then              ShowWindow((Components[J] as
TWinControl).Handle, SW_SHOWDEFAULT);             end;             ShownOnce := True;
end;  end;     procedure TForml.WMSYSCOMMAND(var msg: TMessage);  begin   inherited;
 if (Msg.wParam=SC_MINIMIZE) then      begin       HideMainForm; CreateTraylcon(l) ;     end;
end;     procedure TForml.FileOpenItemlClick(Sender: TObject);  begin     RestoreMainForm;
DeleteTraylcon(l);   end;
```

Теперь у вас в руках полноценный набор средств для работы с панелью System Tray.

В заключение необходимо добавить, что все описанное реализуется не в операционной системе, а в оболочке ОС — Проводнике (Explorer). В принципе, и Windows NT 4/2000, и Windows 95/98 допускают замену оболочки ОС на другие, например **DashBoard** или **Ligh tStep**
. Там функции панели System Tray могут быть не реализованы или реализованы через другие API. Впрочем, случаи замены оболочки достаточно редки.