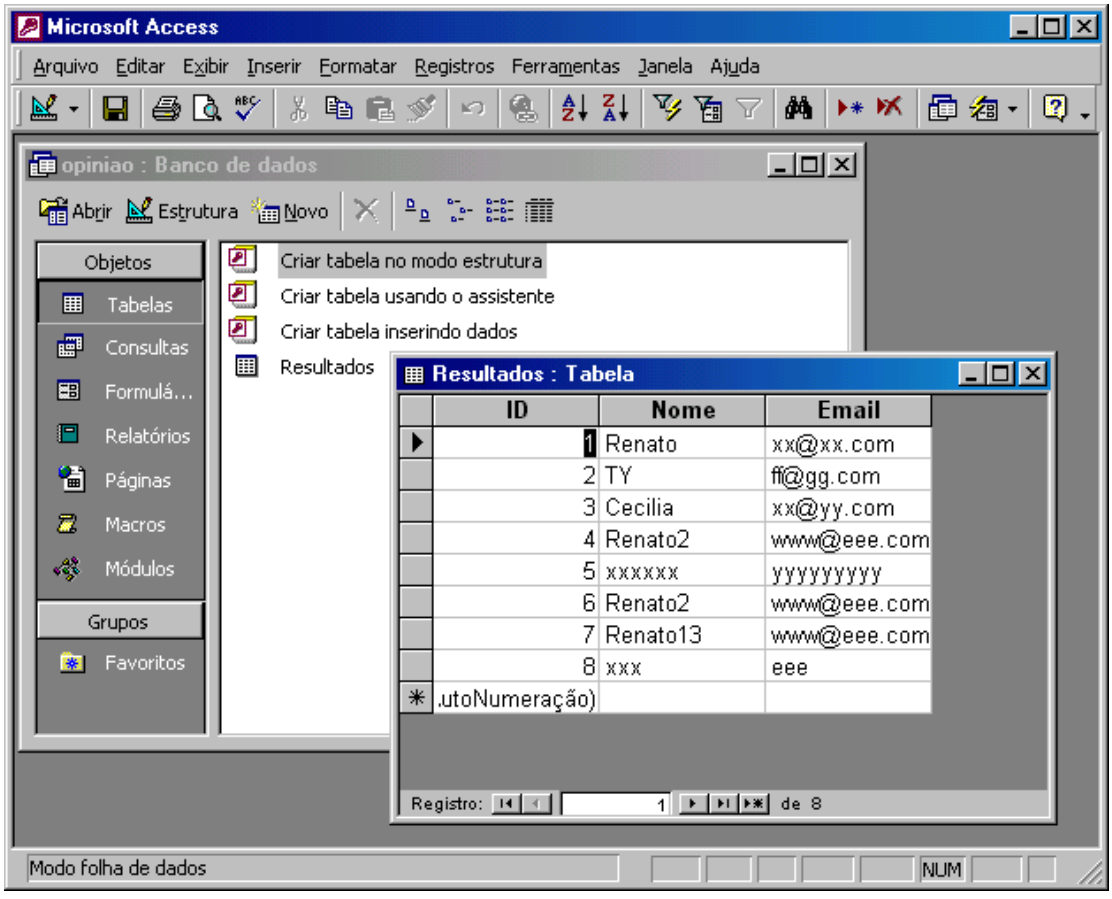
Acessando um Banco de Dados

Introdução

Agora que você já está craque em JSP e já instalou seu servidor, vamos direto para a parte prática!

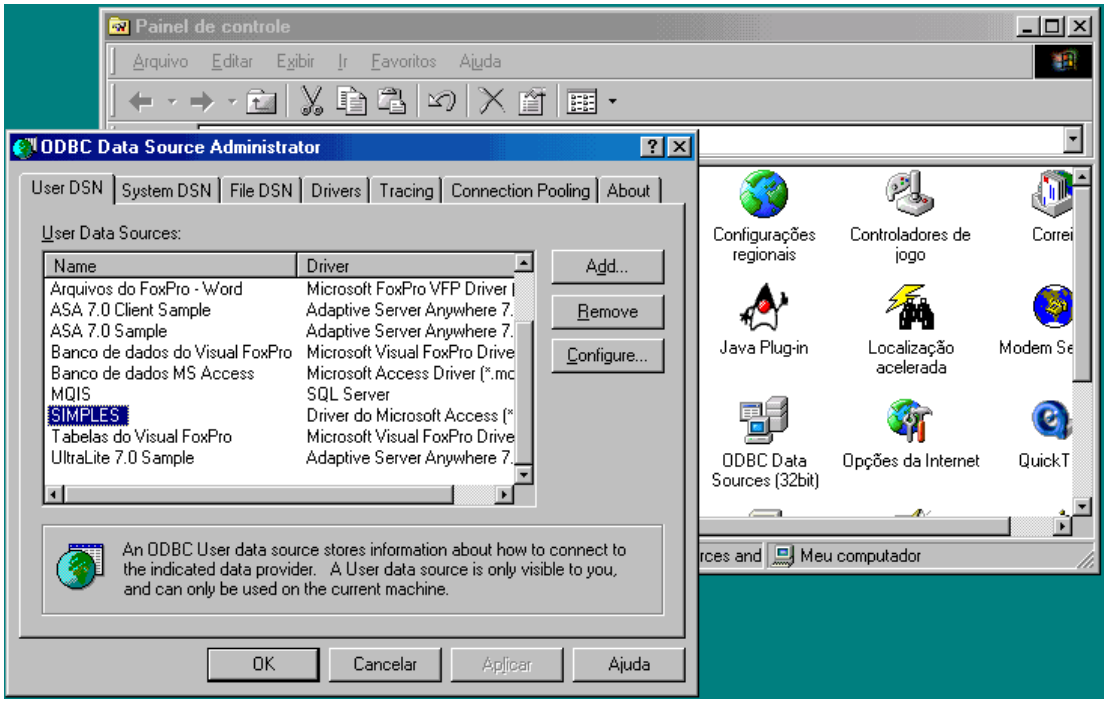
Neste tutorial vamos aprender a acessar um banco de dados. Para quem está acostumado com ASP ou ColdFusion, perceberá que o JSP trata a conexão com a base de dados de uma forma um tanto diferente. Portanto tenha em mente que precisamos montar um sistema que, ao mesmo tempo, exiba formulários de cadastro e pesquisa, e acesse o banco de dados. Temos então uma página chamada `Consulta.jsp` e uma `Incluir.jsp`. Mas estas são as interfaces, uma serve para consultar registros e a outra para incluir registros na base de dados. Mas antes disso devemos criar o banco de dados em si e o objeto que o acessará, ou seja, um servlet bean chamado `aod.class`. Neste servlet estará todo o código que consultará o banco de dados. Vamos por partes!

Configurando a base de dados



Para o nosso exemplo, você precisará de um banco de dados como este acima. Não precisa ser necessariamente um `*.mdb` do Microsoft Access. Você pode usar a base de dados que quiser. Apenas crie uma tabela chamada `Resultados` e os campos `ID`, `Nome` e `Email`. Cadastre alguns registros manualmente para testarmos a consulta daqui a pouco.

Estabelecendo a conexão



Bom, depois de criar o banco de dados, você precisará criar um vínculo com a ponte de dados ODBC, ou seja, você precisa criar um *DataSourceName* (DSN). Note que na figura a tela do ODBC está em inglês. Isto se dá pela utilização do MDAC atualizado, que é necessária principalmente para que o ODBC compreenda arquivos do Microsoft Access 2000. Procure a versão do MDAC mais atual para o seu sistema.

Para criar o DSN, clique duas vezes no ícone *ODBC Data Sources (32bits)* no *Painel de Controle* do Windows. A tela que aparecerá será semelhante a esta da figura acima. Clique em "Add.." ou "Adicionar..." para adicionar um novo DSN. Escolha o driver específico para seu banco de dados.

Na tela seguinte, preencha o campo "Nome da fonte de dados:" com um nome para a sua fonte de dados. No nosso exemplo estamos utilizando o nome "SIMPLES". Digite alguma coisa no campo "Descrição:" se você quiser. Clique no botão "Selecionar..." para configurar o *path* para o banco de dados. Procure pelo arquivo `opiniao.mdb` no seu disco e clique em "OK".

Por último, clique em "Avançado..." e defina um usuário "1234" com a senha "1234". Pronto seu banco de dados está pronto e conectado!

Criando o objeto de acesso a dados

```
// Classe Simples para
// Acesso ao Objeto de Dados (AOD.CLASS)
// Develped by Renato Graccula (http://www.seriallink.com)
// Inspirado no original de Galileu Batista (http://www.jspbrasil.com.br)
package simples;
import java.sql.*;
public class aod extends java.lang.Object {
    // Declaração das Variáveis
    static Connection con = null;
    static private String blank = "";
    private int id;
    private String nome  = blank;
    private String email = blank;
    private int tarefa = 0;
    // Classe Pública AOD
    // Inicia a Conexão ao se criar o objeto
    public aod() {
        if (con == null) {
            try {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                con = DriverManager.getConnection("jdbc:odbc:SIMPLES", "1234", "1234");
            } catch (Exception e) {
                System.err.println ("Erro no comando de Conexao");
            }
        }
    }
    // Método que retorna o campo 'id'
    public int getID() {
        return id;
    }
    // Método que define o campo 'id'
    public void setID(int id) {
        this.id = id;
    }
    // Método que retorna o campo 'nome'
    public String getNome() {
        return nome;
    }
    // Método que define o campo 'nome'
    public void setNome(String nome) {
        this.nome = nome;
    }
    // Método que retorna o campo 'email'
    public String getEmail() {
        return email;
    }
    // Método que define o campo 'email'
    public void setEmail(String email) {
        this.email = email;
    }
    // Método que retorna o campo 'tarefa'
    public int getTarefa() {
        return tarefa;
    }
    // Método que define o campo 'tarefa'
    public void setTarefa(int tarefa) {
        this.tarefa = tarefa;
    }
    // Método para resetar os campos
    public void limpaTudo() {
```

```
        nome = email = blank;
        id = 0;
        tarefa = 0;
    }
    // Método para gravar e pesquisar no banco de dados
    public boolean fazTarefa() {
        // Incluir Registro
        if (tarefa == 2) {
            try {
                // Apenas inclui no banco de dados
                Statement stmt = con.createStatement();
                stmt.execute ("INSERT INTO Resultados (nome, email) VALUES ('" + nome + "', '" +
email + "') ");
                stmt.close();
            }
            catch (Exception e) {
                // Verificar erros
                System.err.println ("Erro no comando SQL de Insert");
                limpaTudo();
                return false;
            }
        }
        // Pesquisar por Nome
        else if (tarefa == 1) {
            try {
                // Faz a pesquisa e retorna um recordset
                Statement stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT * FROM Resultados WHERE Nome = '" +
nome + "' ");
                if (rs.next()) {
                    // Se o recordset contiver dados,
                    // guarda-os nas respectivas variáveis
                    id    = rs.getInt("id");
                    nome  = rs.getString("nome");
                    email = rs.getString("email");
                }
                else {
                    // Recordset vazio
                    System.err.println ("Nenhum Registro!");
                    limpaTudo();
                    return false;
                }
                rs.close();
                stmt.close();
            }
            catch (Exception e) {
                // Verificar erros
                System.err.println ("Erro no comando SQL de Select");
                limpaTudo();
                return false;
            }
        }
        else {
            // Nenhuma Tarefa
            limpaTudo();
            return false;
        }
        return true;
    }
}
```

Neste código acima, temos o grosso da programação Java. Para quem não conhece bem, assusta um pouco, mas não é muito complicado. Basicamente este código, ao ser chamado por um outro servlet – que pode ser um JSP compilado -, cria um objeto que permite acesso a dados. Ou seja, com esse servlet podemos consultar e modificar os registros do banco de dados.

No nosso exemplo, o objeto `aod.class` ao ser chamado em uma página JSP, cria a conexão com o banco de dados. Isso acontece na declaração do método que tem o mesmo nome do objeto:

```
public aod() {
      // código
}
```

Este método é executado no momento que a classe é invocada. Por ser um objeto abstrato, esta classe é chamada de Bean.

Logo após a declaração do método principal – de conexão -, temos os métodos de controle. Ou seja, são os responsáveis pelo acesso e modificação das informações obtidas no banco de dados. O JSP pode perfeitamente fazer uso das variáveis do objeto, utilizando os métodos apropriados.

Método para "pegar" o valor da variável privada do objeto `aod.class`:

```
public getEmail() {
      return email;
}
```

Método para "configurar" o valor da variável privada do objeto `aod.class`:

```
public setEmail(String email) {
      this.email = email;
}
```

Para você entender como esses mecanismos funcionam é só lembrar que o servlet roda junto com o JSP que o importar. Por isso nosso objeto `aod` é invocado por uma página JSP de consulta, por exemplo. E neste momento cria a conexão e torna todos os campos do banco de dados disponíveis nas variáveis do objeto. Mas você deve estar se perguntando: "E isso acontece por acaso?". É uma boa pergunta, mas lógico que não! Isso acontece pois as variáveis do objeto têm o mesmo nome dos campos do banco de dados! E os campos do formulário das páginas JSP também terão o mesmo nome! Este é o truque do JSP! Por essa razão você pode criar métodos bem simples para acessar e manipular os campos do banco de dados.

O método que faz toda essa orquestra funcionar é o `fazTarefa()`. O que esse método faz:

- Executar alguma tarefa?
    - Sim: Tarefa 1 – Pesquisar por nome.
    - Sim: Tarefa 2 – Incluir registro.
    - Não: Nenhuma Tarefa – Limpa variáveis.

Recapitulando, o servlet é invocado pelo JSP. As variáveis são definidas. O método `aod` é executado, criando a conexão com o banco de dados. Nesse exato momento, as variáveis do objeto recebem os valores dos campos do banco de dados. A partir desse momento as informações ficam disponíveis nas variáveis do objeto `aod`.

Bom, da mesma forma que o servlet interage com os campos do banco de dados, o JSP também interage, passando variáveis via POST ou GET. Vejamos o JSP para consultar o banco de dados.

Criando uma página para consultar dados

```
<html>
<head>
    <title>Acesso ao Objeto de Dados</title>
</head>
<body>
<%@ page import = "simples.*" %>
<jsp:useBean id="acesso" class="simples.aod"/>
<jsp:setProperty name="acesso" property="tarefa" param="tarefa"/>
<jsp:setProperty name="acesso" property="nome" param="nome"/>
<%
    int varTarefa = acesso.getTarefa();
    if (varTarefa != 0) {
        acesso.fazTarefa();
%>
        <h1> Resultado de Pesquisa </h1><p>
        Nome: <jsp:getProperty name="acesso" property="nome"/> <br>
        Email: <jsp:getProperty name="acesso" property="email"/> <br>
        <p>
<%
    }
%>
<h1> Formulário de Pesquisa </h1>
<form method=post action=Consultar.jsp>
Nome: <input type=text name=nome> <input type=submit value=Ok>
<input type=hidden name=tarefa value=1>
</form>
</body>
</html>
```

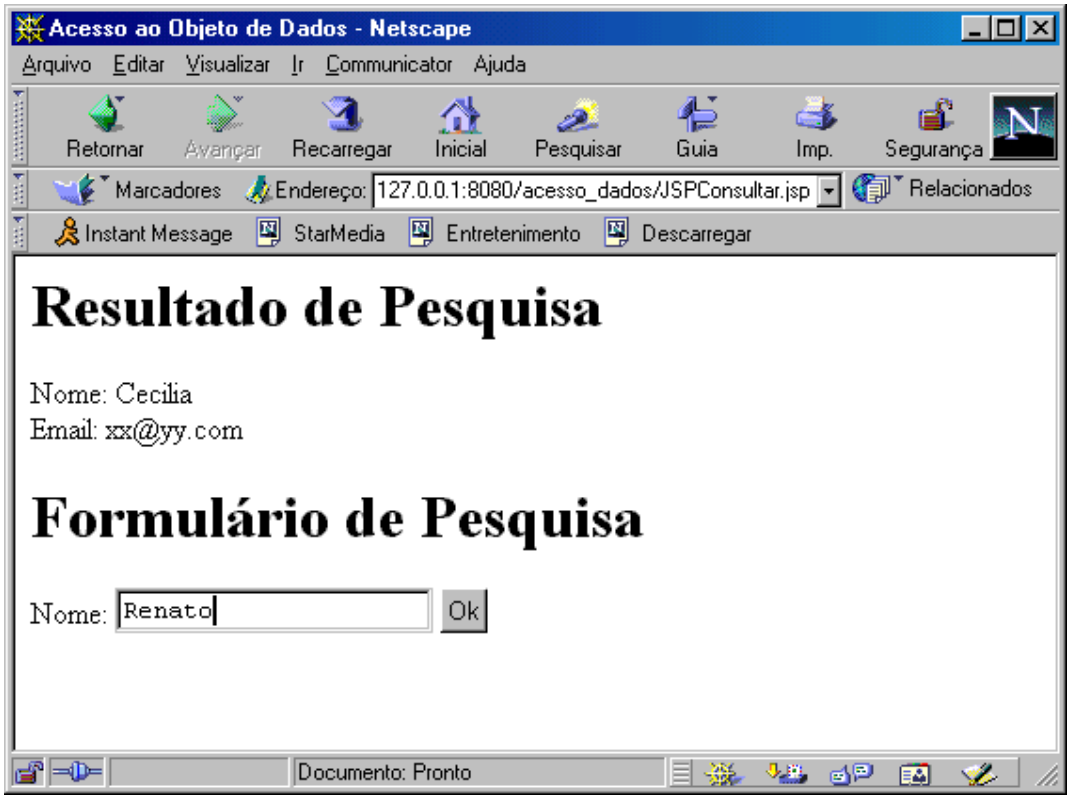
Veja que, esquecendo-se do HTML, a primeira instrução que aparece em Java, é uma tag diretiva page, que tem o atributo import. Essa tag importa todas as classes do pacote simples. Portanto chamando também a classe aod.

Depois de ser importada, a classe aod pode ser usada, como na segunda linha, onde a tag (XML-based) useBean instancia a classe. A partir desse momento, as variáveis estão iniciadas e prontas para uso. Como nenhuma tarefa ainda foi configurada, as variáveis permanecem vazias.

As linhas seguintes (3 e 4) configuram a variável `tarefa` e `nome` para receber os valores passados pela requisição so JSP. Ou seja, na primeira vez que a página é carregada, nenhuma variável é passada como parâmetro, então o Bean, ao ser instanciado, utiliza o valor padrão (zero) para a variável. O mesmo ocorre com a variável `nome`.

Perceba que nas linhas sequintes, do código acima, o método `getTarefa()` é solicitado para verificar se existe alguma tarefa. Se existir, então o método `fazTarefa()` é executado e aparece o "Resultado da Pesquisa" e o "Formulário de Pesquisa". Se o `getTarefa()` retornar zero (quando a página é carregada pela primeira vez), o método `fazTarefa()` não é executado e só aparece o "Formulário de Pesquisa".

O "Formulário de Pesquisa" então completa o cenário. Ao se enviar o formulário, os campos `nome` e `tarefa` são automaticamente incorporados ao Bean (graças as tags `setProperty`). Este então configura a variável `tarefa` para `1`. Então o JSP executa o método `fazTarefa()` e apresenta o "Resultado da Pesquisa". Veja na figura abaixo como deve ser a tela deste JSP.


Acesso ao Objeto de Dados - Netscape
Arquivo Editar Visualizar Ir Communicator Ajuda
Retornar Avançar Recarregar Inicial Pesquisar Guia Imp. Segurança
Marcadores Endereço: 127.0.0.1:8080/acesso_dados/JSPConsultar.jsp Relacionados
Instant Message StarMedia Entretenimento Descarregar
Resultado de Pesquisa
Nome: Cecilia
Email: xx@yy.com
Formulário de Pesquisa
Nome: Renato Ok
Documento: Pronto

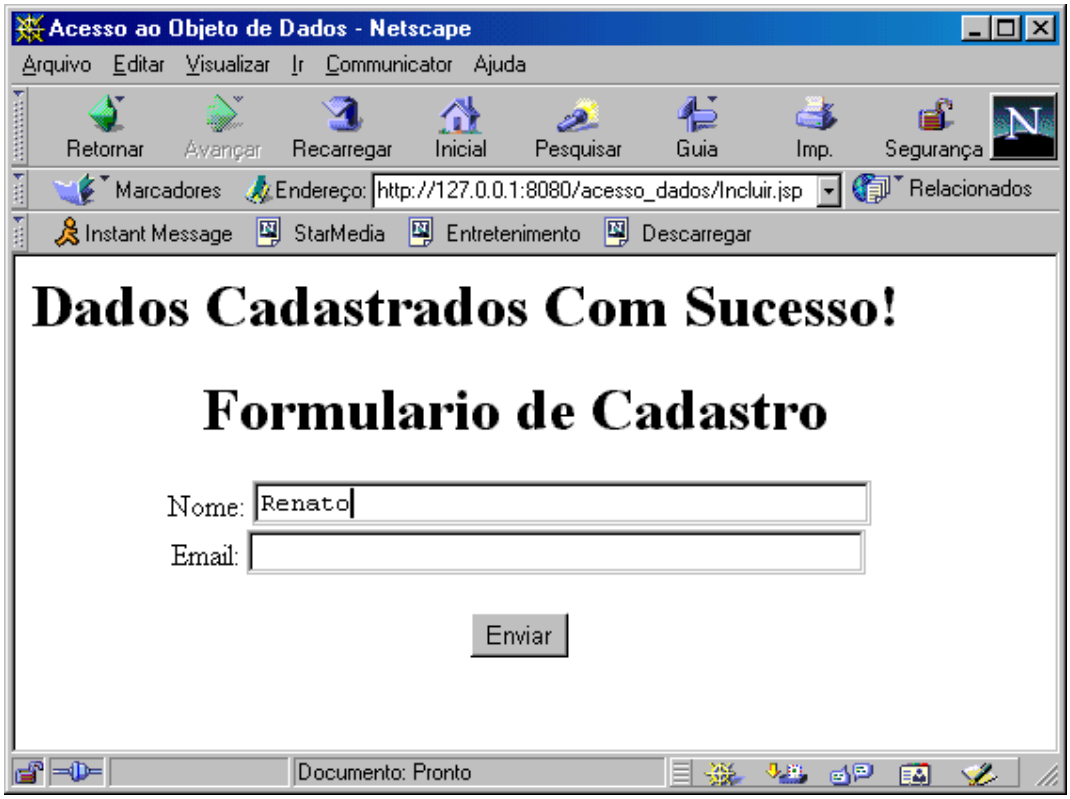
Criando uma página para incluir dados

```
<html>
<head>
    <title>Acesso ao Objeto de Dados</title>
</head>
<body>
<%@ page import = "simples.*" %>
<jsp:useBean id="acesso" class="simples.aod" />
<jsp:setProperty name="acesso" property="tarefa" param="tarefa"/>
<jsp:setProperty name="acesso" property="nome" param="nome"/>
<jsp:setProperty name="acesso" property="email" param="email"/>
<%
    if (acesso.getTarefa() != 0) {
        if (acesso.fazTarefa()) {
            out.println("<H1>Dados Cadastrados Com Sucesso!</H1>");
        } else {
            out.println("<H1>Erro Ao Cadastrar Dados!</H1>");
        }
        out.println("<p>");
    }
%>
<center>
<H1> Formulario de Cadastro </H1>
<form method=POST action="Incluir.jsp">
Nome: <input type=text name="nome" size=40> <br>
Email: <input type=text name="email" size=40> <br>
<p>
<input type=hidden name=tarefa value=2>
<input type=submit value=Enviar>
</form>
</body>
</html>
```

O código para incluir registros no banco de dados não é muito diferente do que vimos anteriormente. As principais diferenças são:

- A variável `tarefa` é configurada para 2, para executar a porção de código que inclui o registro no banco de dados.
- A tag `setProperty` agora configura também o campo `email`, para que este seja passado para a variável `email` dentro do objeto `aod`.
- Dependendo do resultado do método `fazTarefa()`, o JSP imprime se a tarefa foi bem sucedida ou não.

Ao ser executado pela primeira vez o JSP apresenta apenas o formulário. Quando o form é acionado, os campos `nome`, `email` e `tarefa` serão passados como parâmetros e as variáveis do objeto `aod` recebem os valores. Com isso o JSP executa o método `fazTarefa()` que é o responsável por inserir e pesquisar registros. Como nesse caso, o campo `tarefa` foi passado para a variável `tarefa` dentro do Bean, com o valor 2, a tarefa executada é a inclusão das variáveis `nome` e `email` no banco de dados, respectivamente importadas dos campos `nome` e `email` do formulário da página JSP. Veja na figura abaixo como deve ser a tela deste JSP.



/------
/ Tutorial retirado da Serial Link Millenium 3000
/ www.seriallink.com
/ Desenvolvido por Serial Link
/------